SALIENT FEATURE SELECTION USING FEED-FORWARD NEURAL NETWORKS AND SIGNAL-TO-NOISE RATIOS WITH A FOCUS TOWARD NETWORK THREAT DETECTION AND CLASSIFICATION

THESIS

Kristy L. Moore, Major, USAF

AFIT-ENS-14-M-22

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT-ENS-14-M-22

SALIENT FEATURE SELECTION USING FEED-FORWARD NEURAL NETWORKS AND SIGNAL-TO-NOISE RATIOS WITH A FOCUS TOWARD NETWORK THREAT DETECTION AND CLASSIFICATION

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Kristy L. Moore

Major, USAF

March 2014

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT-ENS-14-M-22

SALIENT FEATURE SELECTION USING FEED-FORWARD NEURAL NETWORKS AND SIGNAL-TO-NOISE RATIOS WITH A FOCUS TOWARD NETWORK THREAT DETECTION AND CLASSIFICATION

Kristy L. Moore
Major, USAF

Approved:

//signed//	25 Feb 2014
Dr. Kenneth W. Bauer, Jr. (Chairman)	Date
//signed//	25 Feb 2014
Thomas E. Dube, Maj, USAF (Member)	Date
//signed//	25 Feb 2014
David M. Ryer, Lt Col, USAF (Member)	Date

AFIT-ENS-14-M-22

Abstract

Most communication in the modern era takes place over some type of cyber network, to include telecommunications, banking, stock exchanges, vehicular traffic flow, public utilities, health systems, and social networking to name a few. Information gained from illegitimate network access can be used to create catastrophic effects at the individual, corporate, national, and even international levels, making cyber security a top priority.

Cyber networks frequently encounter amounts of network traffic too large to process real-time threat detection efficiently. Reducing the amount of information necessary for a network monitor to determine the presence of a threat would likely aide in keeping networks more secure.

This thesis uses network traffic data captured during the Department of Defense Cyber Defense Exercise to determine which features of network traffic are salient to detecting and classifying threats. After generating a set of 248 features from the capture data, feed-forward artificial neural networks were generated and signal-to-noise ratios were used to prune the feature set to 18 features while still achieving an accuracy ranging from 83% - 97% for the testing/training sets and 63% - 88% for the validation sets. The salient features primarily come from the transport layer section of the network traffic data and involve the client/server connection parameters, size of the initial data sent, and number of segments and/or bytes sent in the flow.

Dedication

To my husband and son who make it all worth it.

Acknowledgments

I would like to express my sincere gratitude to my faculty advisor, Dr. Kenneth Bauer, for his guidance and support and for allowing me to explore a topic that extends beyond the realm of Operations Research. I would also like to thank my reader, Maj Thomas Dube, for his insights and assistance and Dr. Timothy Lacey for providing technical support in the development and processing of the datasets. I would also like to express my appreciation to CPT James Jablonski for his help with the MATLAB code and to Trevor Bihl for his technical expertise and insights.

Finally, and most importantly, I would like to thank my family for their love, support, encouragement, and sacrifice throughout this process. Thank you for putting up with me and for always believing in me.

Kristy L. Moore

Table of Contents

Page

List of Figures

Page

List of Tables

Page

SALIENT FEATURE SELECTION USING FEED-FORWARD NEURAL NETWORKS AND SIGNAL-TO-NOISE RATIOS WITH A FOCUS TOWARD NETWORK THREAT DETECTION AND CLASSIFICATION

I. Introduction

1.1. Background

The integration of cyber technologies into nearly all aspects of our everyday lives makes cyber security a serious concern. Cyber security has been defined as a "complicated and complex subject encompassing computer security, information assurance, comprehensive infrastructure protection, commercial integrity, and ubiquitous personal interactions" [1]. Most communication in the modern era takes place over some type of cyber network, to include telecommunications, banking, stock exchanges, vehicular traffic flow, public utilities, health systems, and social networking to name a few. Information gained from illegitimate network access can be used to create catastrophic effects at the individual, corporate, national, and even international levels. The same could be said for successfully executed attacks against those networks. The number of cyber attacks on Department of Defense (DoD) and other United States (US) Government networks is estimated to be 400 million annually [2]. A study published by McAfee Security suggests US losses due to cyber attacks may reach $100 billion per year [3]. Cyber security has become important enough to be listed as one of the five central missions of the Department of Homeland Security [4] and FBI officials speculate cyber attacks will surpass terrorism as a domestic danger over the next ten years [5].

1.2. Research Goal and Objectives

An important aspect of cyber security is threat detection and classification. The aspect of cyber security chosen for this research is computer network traffic. Computer network traffic encompasses a massive amount of information. While all of that information can be captured fairly easily (digital storage space is relatively cheap), analysis of the information is a time and resource intensive process. The issue becomes how to sort through the information to determine what is a threat and what is not.

The purpose of this research is to reduce network traffic data into the parts, or features, salient to threat detection and classification. The inspiration for this thesis comes from Moore et al. [6], a research study conducted in 2005, to develop a set of features to assess classification performance on general network traffic data. The outcome of the study was a list of 248 features based on packet or flow data. After generating a dataset consisting of those 248 features from a collection of known network attack data, this thesis attempts to determine which of the features are most important to determining whether or not a threat exists.

1.3. Assumptions and Limitations

One of the challenges faced when attempting to classify computer network threats is acquiring the truth data. The people with malicious intent toward a network are unlikely to give away information on their exploits making it difficult to label threats. The data used for this research came without truth data making it necessary to derive a way to create labels for each observation. The method used for this label creation is discussed in Chapter III; it is assumed the observations using this method are labeled accurately.

1.4. Implications

Much research has gone into defining the salient features for general traffic classification, some of which will be discussed in Chapter II. The research in this thesis intends to narrow that focus down, specifically targeting threat detection and classification. The hope is that most of the data will be unnecessary and can be disregarded while still capturing the pertinent information. Ideally, this would be done in real-time with a network sensor that monitors incoming traffic and extracts only the data necessary for detecting and classifying threats. Successful reduction of the dataset required for analysis, along with knowledge of what components of the network traffic to focus on, should noticeably speed up the threat detection process and potentially enable a more secure computer network.

1.5. Preview

Chapter II presents previous work done in the areas of general network traffic and network threat detection classification. It also includes a discussion of the use of neural networks for classification and the use of the signal-to-noise ratio as a saliency measure. Chapter III describes the methodologies used in experimentation of the datasets including the work involved in preprocessing the data for analysis. Chapter IV explores the results and analysis from the experimentation done with the data. Finally, Chapter V discusses the conclusions developed from the results in Chapter IV and offers thoughts on future research of this topic.

II. Literature Review

2.1. Chapter Overview

With the explosion of computer networks and internet usage across the world in the last 15 to 20 years, methodologies to better understand and classify internet traffic have become a hot research topic for a multitude of reasons. This chapter provides a summary of previous work done relevant to the research presented in this thesis and is organized as follows. The first section attempts to define some of the more common terms and concepts to help the reader better understand the typical technical jargon. Next, we will examine research in the areas of general internet traffic classification methods to gain some insight into the overarching methodologies, and then focus in on the sub-field of network intrusion detection classification methods, which relates directly to the research in this thesis. Finally, we will consider research done on the importance of feature selection to accurate classification, and, while there are many methods to handle feature selection, we will focus in on using neural networks, leading up to the method used in this thesis.

2.2. Background Information

The research in this thesis attempts to merge together two distinct, yet related, academic fields, operations research (OR) and cyber operations (specifically computer networks). As such, terminology and concepts may be used differently between the two. Because this is an OR-based thesis, this section is an attempt to bridge the gap between the two academic fields, making it easier for the OR-based reader to better understand both the reviewed research and the research presented in this thesis. We will begin by defining some of the more commonly used terminology and concepts to provide a

baseline understanding. Next is a brief discussion of the currently used TCP/IP model used for describing how computer networks work. An acronym list can be found in Appendix A: Acronym List.

2.2.1. Terminology and Concepts

The following are explanations of terminology and concepts seen throughout the reviewed literature and the research done for this thesis. Most of the definitions come from Technopedia.com [7], although a few are derived from experience or noted sources.

- *Audit:* an examination of a computer network's traffic logs or administrative policies and procedures [8].
- *Bandwidth:* broad term defined as the bit-rate measure of the transmission capacity over a network communication system. Bandwidth is also described as the carrying capacity of a channel or the data transfer speed of that channel. However, broadly defined, bandwidth is the capacity of a network. Bandwidth exists in both the wired and wireless communication networks.
- *Client:* can be a simple application or a whole system that accesses services being provided by a server; most often located on another system or computer, which can be accessed via a network.
- *Client/Server Architecture:* a computing model, in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or Internet connection. This system shares computing resources.

- *Computer Networks:* a group of computer systems and other computing hardware devices that are linked together through communication channels to facilitate communication and resource-sharing among a wide range of users.
- *Encryption:* the process of using an algorithm to transform information to make it unreadable for unauthorized users.
- *Firewalls:* software, hardware, or a combination of both, used to maintain the security of a private network. Firewalls block unauthorized access to or from private networks and are often employed to prevent unauthorized Web users or illicit software from gaining access to private networks connected to the Internet.
- *Flows:* one or more packets between a pair of hosts, defined by a 5-tuple, made up of source and destination IP addresses, source and destination port numbers, and the protocol type (TCP, UDP) used for communication.
- *Header:* the initial set of bits in a packet transmitted by an end device that describes what the receiving end device can expect to receive throughout the data stream.
- *Hosts:* end systems, sometimes referred to as clients or servers [9].
- *Hypertext Transfer Protocol (HTTP):* an application-layer protocol used primarily on the World Wide Web. HTTP uses a client-server model where the web browser is the client and communicates with the webserver that hosts the website. The browser uses HTTP, which is carried over TCP/IP to communicate to the server and retrieve Web content for the user.
- *Internet Protocol (IP):* protocol that specifies the format of the packets sent and received among routers and end systems [9]. One of the two most important

protocols in the Internet; all internet components with network layers must use IP. The most common version is IPv4, although IPv6 may take over in the future. IPv4 is a connectionless protocol providing the logical connection between network devices by providing identification for each device [7].

- *Layer:* a logical grouping of similar functions; used in networking to distinguish the communication functions associated with computer networks [10].
- *Network Planning and Resource Provisioning:* analyzing network traffic/behavior to allocate resources to optimize prioritization and performance.
- *Network Security Monitoring:* a computer network's systematic effort to detect, deter and track unauthorized access, exploitation, modification, or denial of the network and network resources.
- *Network Traffic:* the flow of data across a computer network.
- *Offline vs. Online:* online refers to analysis or classification done in real-time or near real-time while the system is monitoring the network and collecting data; offline refers to analyzing or classifying data that has already been collected.
- *Packet:* a single network communication data unit containing fixed or variable lengths, and may contain three portions: header, body and trailer.
- *Payload:* the raw data a packet carries.
- *Ports:* process-specific or application-specific software construct serving as a communication endpoint. A specific network port is identified by its number commonly referred to as port number, the IP address in which the port is associated with and the type of transport protocol used for the communication. Any networking process or device uses a specific network port to transmit and

receive data. This means that it listens for incoming packets whose destination port matches that port number, and/or transmits outgoing packets whose source port is set to that port number. Processes may use multiple network ports to receive and send data.

- *Protocols:* a set of rules and guidelines for communicating data. Rules are defined for each step and process during communication between two or more computers. Networks have to follow these rules to successfully transmit data.
- *Routers:* a device that analyzes the contents of data packets transmitted within a network or to another network. Routers determine whether the source and destination are on the same network or whether data must be transferred from one network type to another, which requires encapsulating the data packet with routing protocol header information for the new network type.
- *Quality of Service (QoS):* refers to a network's ability to achieve maximum bandwidth and deal with other network performance elements like latency, error rate and uptime. Quality of service also involves controlling and managing network resources by setting priorities for specific types of data (video, audio, files) on the network.
- *Server:* a computer or computer program that manages access to a centralized resource or service in a network [11].
- *Three-Way Handshake:* a method used in a TCP/IP network to create a connection between a local host/client and server. It is a three-step method that requires both the client and server to exchange SYN (synchronization),

SYN/ACK, and ACK (acknowledgment) packets before actual data communication begin.

- *Transmission Control Protocol (TCP):* a network communication protocol designed to send data packets over the Internet. The other of the two most important protocols in the internet. TCP is a transport layer protocol used to create a connection between remote computers by transporting and ensuring the delivery of messages over supporting networks and the Internet. TCP works in collaboration with IP, which defines the logical location of the remote node, whereas TCP transports and ensures that the data is delivered to the correct destination. Before transmitting data, TCP creates a connection between the source and destination node and keeps it live until the communication is active. TCP breaks large data into smaller packets and also ensures that the data integrity is intact once it is reassembled at the destination node
- *Tunneling: a* protocol enabling the secure movement of data from one network to another. Tunneling uses an encapsulation process to make data packets appear as though they are of a public nature to a public network when they are actually private data packets, allowing them to pass through unnoticed. Examples of tunneling include Virtual Private Networks (VPN) and Hypertext Transfer Protocol (HTTP).
- *User Datagram Protocol (UDP):* transport layer protocol for client- server network applications. UDP does not employ handshaking dialogs for reliability, ordering and data integrity. The protocol assumes that error-checking and correction is not required, thus avoiding processing at the network interface level.

UDP is widely used in video conferencing, real-time computer games, and data streaming. The protocol permits individual packets to be dropped or received in a different order than that in which they were sent, allowing for better performance.

- *Webserver:* a system that delivers content or services to end users over the Internet. A Web server consists of a physical server, server operating system (OS) and software used to facilitate HTTP communication.

2.2.2. TCP/IP Model

In 1984 the Open Systems Interconnection (OSI) Model was published by the International Organization for Standardization (ISO) to provide a conceptual model that defines networking standards for hardware and software technology development and how networking protocols should work [10]. As can be seen in Figure 2.1, the model breaks down similar functions into logical (the ways the functions act as opposed to physical placement) layers. The more currently used model, the TCP/IP model, was developed after the OSI Model around the TCP/IP protocols, what we now call the internet [12]. The TCP/IP model essentially combines layers from the OSI model into broader categories more appropriate to current computer networking. The model is sometimes seen with five layers (separating the network access layer into the data link and physical layer). The most pertinent information for the reader to know is that the research for this thesis is focused mostly in the transport layer which handles end-to-end connections, and is the same in both modes, with some interaction in the internet/network layer.

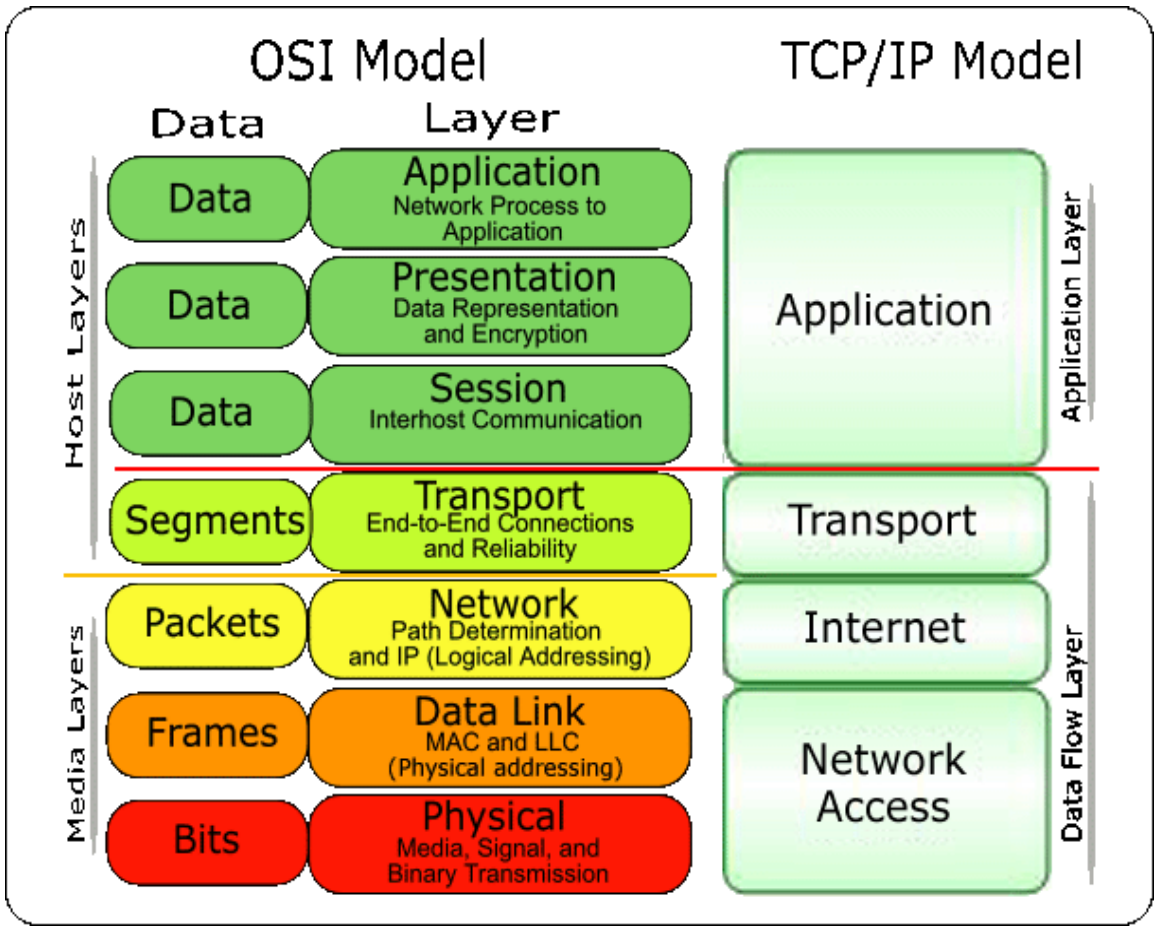

Figure 2.1: OSI vs. TCP/IP Model [13]

2.3. General Internet Traffic Classification Methods

Network intrusion detection classification is a subset of general internet traffic classification and, as such, the overall classification process and its methodologies are the building blocks for the techniques used in intrusion detection classification. General traffic classification is imperative in network planning estimation and resource provisioning, security monitoring and auditing, and Quality of Service (QoS) measurements. Classification methods discussed here include transport layer port analysis and payload inspection (signature rule-based matching), machine learning with both unsupervised and supervised algorithms, and a few other, uncategorized methods. Many

of the techniques discussed here can also be found in surveys done by Nguyen and Armitage [14] and Dainotti et al. [15].

2.3.1. Port Analysis and Signature-Based Methods

Classifying network applications based on well-known port numbers is considered by some as the simplest, and fastest, method of classification, as long the classification accuracy is not vital [15]. Moore and Papagiannaki [16] explain that, as technology and user skills have developed and adapted, well-known port numbers are no longer reliable for use in classification. This is due mainly to internet applications being designed to use other than standard port numbers, dynamic port selection, or protocols as wrappers to slip through security systems, like firewalls, unnoticed. Moore and Papagiannaki show this change in design leads to a low accuracy classification (50% - 70%) of network traffic when using port-based classification from the Internet Assigned Numbers Authority (IANA) list [17]. The authors present a classification technique using a content-based (payload) methodology. This classification technique examines and interprets the contents of the packet's payload iterating through nine methods including port-based classification, packet header information, single packet signatures, single packet protocols, first kilobyte signatures, first kilobyte protocols, selected flow protocols, all flow protocols, and host history. Each method is applied sequentially until the required classification certainty has been reached. While this technique achieves an impressive overall average accuracy of 98%, it is obviously quite labor intensive.

Roughan et al. [18] suggest classifying network traffic through the use of statistical application signatures. These signatures are derived from the manner in which the applications are used, forming a set of classification rules based on port numbers or IP

addresses, and are intended to be insensitive to the particular application layer protocols. The technique first calculates connection statistics offline, using those results to classify the traffic, then creates rules for the online classifier to use. Nearest Neighbor and Linear Discriminant Analysis were the two methods used for classification. Using trial-and-error, the authors selected up to four features to classify the data across four to seven application classes. Although encouraging, because their analysis is restricted to broadly defined classes, the authors limit their resulting low error rates (5% - 8%) to wide-ranging properties found in many applications, meaning the classification method will be insufficient for identifying specific applications.

Haffner et al. [19] proposes using application-level information taken from packets to match with common application signatures. Three machine learning (ML) classifiers are used to derive the application signatures for several network applications – Naïve Bayes, AdaBoost, and Maximum Entropy. The classifier intends to be insensitive to port numbers, alterations of network characteristics, and communication pattern changes. This method of matching application signatures does, however, require frequent updating because the application signatures are dynamic and may change over time with application and protocol evolvement. The AdaBoost algorithm results in the best classification performance (greater than 99%); the authors suggest this may be due to the extremely low noise level in the data because of the way it was generated. The research also considered early classification resulting in the conclusion that only the first 64 bytes of each flow is necessary for application identification. Finally, the authors test the derived signatures against data captured seven months later, finding only a slight increase in the error rate, demonstrating the durability of the signatures over time.

2.3.2. Machine Learning (ML) Methods

There are two standard categories of machine learning methods: Unsupervised and Supervised. Unsupervised learning means the correct classification labels are not provided with the data [20]. The underlying structure of the data is examined, looking for correlations in the data to discern patterns, which are then organized into categories. Supervised learning requires the correct classification labels with the data. Weights are then used to help the generated network achieve classification as close to the correct answers as possible. The next few sections discuss clustering, a type of unsupervised machine learning, and several different types of supervised machine learning methods.

2.3.2.1. Unsupervised Machine Learning Methods

Obstacles such as privacy information, encryption, and protocol tunneling (encapsulation) have made it difficult, if not impossible, to inspect the payload, or data, carried in the packets across a network. This, along with increased complexity and processing overhead, has moved the focus of classification techniques away from payload inspection and introduced the concept of ML, incorporating unsupervised and supervised algorithms [15]. This section discusses different uses of the unsupervised ML algorithm referred to as clustering.

McGregor et al. [21] propose the probabilistic Expectation Maximization (EM) algorithm to designate flows into clusters based on a fixed set of traffic flow statistics. This classifier looks for similar properties but provides no explanation as to why the applications are grouped the way they are; it may, however, provide some insight with previously unclassified, unknown traffic [22]. McGregor et al. [21] conclude their

clustering technique, at this point, is too general to classify individual applications and new attributes must be developed to illuminate distinctions between those applications.

Zander et al. [23] use AutoClass, a Bayesian clustering algorithm based on EM, in their approach to traffic classification. The feature selection technique for their methodology is based on the classification performance of the AutoClass algorithm. Using sequential forward selection (SFS), also called stepwise selection, features are added one at a time until the best performing combination is derived. The authors calculate the homogeneity of the classes, illustrating how an application with a higher homogeneity is more likely to be separated because its characteristics are dissimilar to the other applications. One issue with this classification methodology is that some applications, such as FTP, Telnet, and Web traffic, overlap each other or have a wide range of characteristics thus making them difficult to separate into classes. While the average classification accuracy of this technique over all given applications is 86.5%, this diversity of characteristics results in a false positive rate of up to 40%, depending on the application.

Unsupervised clustering is also used by Erman et al. [24] for traffic classification, this time comparing the K-Means (partition-based) and DBSCAN (density-based) algorithms with the previously used AutoClass (probabilistic model-based) algorithm. While the DBSCAN algorithm results in a lower overall average classification accuracy (75%) than K-Means and AutoClass (both greater than 85%), it is noted that DBSCAN's clusters are more accurately formed. The model building time of the two newer algorithms (K-Means, one minute; DBSCAN, three minutes) demonstrates a significant difference when compared with the old (AutoClass, four and a half hours). This research

suggests the K-Means algorithm as the best option because of its high classification accuracy and low model building time.

Bernaille et al. [25] focus on analyzing only the first five packets of the TCP flow in an attempt to develop an online, near real-time classifier. Offline traces are used to train the classifier based on the size of the data packets using Euclidean distance. K-Means clustering is used to find natural clusters in the data, chosen because this method does not rely on previously defined classes. This lack of reliance allows for applications with multiple behaviors to be modeled separately. The description and composition of each cluster determine how the online classifier identifies the traffic flow. This method achieves an average accuracy of greater than 80% for the flows tested. One potentially serious limitation exists with this online classifier, however, if the network monitor uses packet sampling instead of complete packet capture because the technique requires the first five packets of the flow.

2.3.2.2. Supervised Machine Learning Methods

This section discusses several Supervised ML algorithms and their application to internet traffic classification. These techniques include Naïve Bayes (NB), C4.5 Decision Tree (C4.5), k-Nearest Neighbor (kNN), Support Vector Machines, and Neural networks.

Moore and Zuev [22] demonstrate internet traffic classification through the NB method, both with and without kernel density estimation. Kernel density estimation is a non-parametric (infinite-dimensional) method of estimating the probability density function [26]. Feature selection is addressed in this research by use of the Fast Correlation-Based Filter (FCBF) [22]. Straight NB resulted in an average classification accuracy of 65.26%; using FCBF pre-filtering brought that value up to an average of

94.29%. NB with kernel density estimation demonstrated an average accuracy of 93.5%, with an increase to average of 96.29% after using FCBF pre-filtering. The experiment was repeated with data from approximately 12 months after the initial data set resulting in a severe drop in classification accuracy for the NB method with an overall average accuracy of 20.75% without kernel estimation and 37.65% with kernel estimation. Using the FCBF pre-filtering brings those values up to 93.38% and 93.73%, respectively, demonstrating the value of the dimension reduction of the features used for analysis.

Williams et al. [27] compared five ML algorithms for internet traffic classification on the basis of both classification accuracy and computational performance. The authors illustrated how classification accuracies can be very similar between the different algorithms while computational performance can be significantly different; this is very important when considering real-time analysis. The five supervised machine learning algorithms analyzed were NB (both discretisation (NBD) and kernel density estimation (NBK)), C4.5, Bayesian Network (BayesNet), and Naïve Bayes Tree (NBTree). The same 'full feature set' (containing 22 features) is used for each ML algorithm tested. Consistency-based (CON) and Correlation-based (CFS) algorithms are then used for feature selection or reduction and the tests are re-run for comparison. There is little change in the classification accuracy (2 – 2.5%) across the five algorithms tested when comparing the full and reduced feature sets. NBK is the only algorithm not at or above the 95% accuracy level (~80%). With computational performance C4.5 has the fastest classification speed, with NBK the slowest, and NBK has the fastest build time, with NBTree the slowest.

Zhang et al. [28] take the NB method a step further by using it in combination with a bag-of-flows (BoF), or a correlated grouping of flows occurring in the same time period. The BoF concept allows for the correlated flows to be aggregated which, when used in concert with the NB algorithm and a set of combination decision rules (sum, maximum, median, and majority), create a set of posterior probabilities used for class prediction. Referred to as BoF-NB, the proposed classification method is then compared to four other classification methods including C4.5, k-Nearest Neighbor (k-NN), NB, and a semi-supervised method proposed by Erman et al. [29]. Six out of the original 20 features are selected through CFS. Results demonstrate the BoF-NB performed, in general, as good as or better than the other methods tested with an overall average accuracy of 88% to 94%, depending on the data set chosen. The authors believe the better performance is because the BoF-NB's effective use of the flow correlation information.

Huang et al. [30] suggested using k-NN in their classification model. To achieve the best classification results the authors select 10 features and 6 classes, based on performance of the classifier. Mahalanobis, in contrast to Euclidean, distance was used to measure the distance between the data samples. This research focused on the change in classification accuracy as additional classes were added, allowing for finer grained classification. Using only three of the classes results in classification accuracies of greater than 99%; however, once the additional three classes are added in, classification accuracies drop as low as 46%. According to the authors, these results occur because the selected classes have similar statistical features and, with the selected features, the correlation between the classes is not taken into consideration.

In an extension of [22], Auld et al. [31] demonstrated a traffic classifier using supervised ML based on Bayesian trained neural networks without source or destination host address or port information, enabling classification of anonymized or encrypted packet headers. Unlike [22], the classification methodology does not assume independence between the discriminators, allowing for a more robust and useful classifier. This experiment consists of data collected from 2 24-hour periods separated by 8 months and focuses solely on TCP flows (i.e., ignores UDP, ICMP, etc.). Multilayer perceptron classification networks, containing one hidden layer with 10 nodes and Bayesian-inferred weights, assign classification probabilities to the traffic flows using 246 features as inputs. This research used the hyperbolic tangent function as the activation function to model nonlinearities. Using the neural networks, the experiments resulted in an average classification of greater than 99% for the first data collection period and 95% for the second. Comparisons are also studied in consideration of the data sizes of the different classes of traffic and its effect on accuracy.

Expanding upon both [22] and [31], Zhou et al. [32] propose a traffic classification technique using feed-forward neural networks trained through Bayesian regularization and compare its results to the NB classifier subject to the Gaussian distribution assumption. The neural networks used for this research are set up very similarly to those in [31] with a single hidden layer containing 10 neurons. Results of the experimentation demonstrate the feed-forward neural networks, with an average overall accuracy of about 95%, perform better than the NB classifier, with an average overall accuracy of about 75%, and are more stable. The authors argue the NB classifier's poor

performance is due to a misrepresentation of the traffic flow properties because of the Gaussian distribution assumption.

Much of the internet traffic classification research focuses on offline classification due to heavy processing overhead and potential bandwidth limitations. Li and Moore [33] propose an online, near-real time, classification system using packet-header based behavioral features and C4.5. This technique requires only information readily available to internet routers without reliance on port numbers or payload inspections and enables examination of latency and throughput of the monitoring system. Because this methodology is designed to be near real-time, the approach capitalizes on features pulled from only an initial few packets (5 – 10), as opposed to those based on entire flows. A CFS method was used for feature dimension reduction leading to a selection of 12 features. C4.5 averaged an overall average classification accuracy of 99.834%. This research also shows, for this methodology, using part of the traffic flow, instead of the entire flow, does not reduce the classification accuracy.

Based on a trial-and-error approach, Tabatabaei et al. [34] choose seven as the necessary number of packets required for online traffic classification comparing Support Vector Machines (SVM) and k-NN techniques. SVM is typically used as a binary classifier so several binary classifiers are combined to create a multi-category SVM focusing on “fuzzy” one-against all and “fuzzy” pairwise techniques. The authors handle feature selection through a minimum redundancy-maximum relevance (MRMR) technique based on maximum statistical dependency, choosing 40 flow and packet-based features. The three classification techniques are compared using both the complete traffic flow and just the first seven packets resulting in the highest average accuracy of 84.9%

occurring with the SVM fuzzy one-against-all using only the first seven packets. For all techniques, using only the first seven packets shows a higher average accuracy; the authors explain this may be because the first few packets contain the setup parameters and those parameters distinguish the different applications.

The Bag-of-Words (BoW) ML classification model proposed by Zhang et al. [35] is designed to use application categories as a representation of the bags and centroids to represent the words. The authors create BoW vectors consisting of the size of the first five packets, source port, destination port, and transport-layer protocol; then cluster them to create the centroids. Vectors representing traffic categories are constructed and the nearest neighbor algorithm, along with the cosine similarity, calculated between the training representation vectors and the incoming flow representation vectors, classifies the incoming traffic. A novel consideration of this research is the effect of out-of-order packet arrival. BoW technique results are compared to those of C4.5 algorithm using both in-order and out-of-order packet data. The BoW methodology scores remain stable at 88.4% while C4.5 drops from 87.15% to 78.33% when the out-of-order data is used. The authors explain this is because the BoW technique does not preserve any order information preventing it from having any effect on the classification accuracy.

2.3.3. Other Classification Methods

Some classification methods fail to fall neatly into one of the above categories. The methods still provide valuable insight into the focus of this thesis and, thus, should be included so they are gathered together here. Crotti et al. [36] present a classification algorithm built around normalized anomaly thresholds and ‘protocol fingerprints’ consisting of three features (packet size, inter-arrival time, and arrival order) from

captured traffic packets. Unlike the majority of the other classification techniques, authors designed this technique to be site-dependent, or not transportable, meaning each site needs its own fingerprints. A series of Probability Density Functions (PDFs) describe the behavior of the packets for certain protocols. Classification is done by statistically matching the behavior of the traffic flow with one of the PDFs, creating an anomaly score based on how far the flow is from the chosen PDF. Anomaly thresholds indicate the highest score a flow can have to be considered a member of a certain protocol; the smaller the threshold, the more accurate the classifier. The authors limit their testing to just three protocols (HTTP, POP3, and SMTP), looking at only the first four packets from each flow, achieving an average classification accuracy to around 91%.

Li and Kianmehr [37] apply a classification methodology based on associative classifiers. Associative classifiers combine associative rule mining, or pattern/correlation searches, with classification with the intention of creating classification models that are easier for users to understand than previously studied ML algorithms. Essentially, the rules are first generated from the training set then pruned to derive the best set of rules. The classifier is built from that best set of rules. The three associative classification algorithms compared by the authors are Classification-Based Association (CBA), Classification-Based on Multiple Association Rules (CMAR), and Classification-Based on Predictive Association Rules (CPAR). Feature selection is handled through an embedded version of chi-squared that computes a statistic with respect to class and uses it to determine the value of the feature. The results show the CPAR algorithm performs the best out of the three tested with an overall average accuracy of 92.05%.

2.4. Network Threat Detection Classification Methods

The earlier section presented previous work related to the broad category of general network traffic classification. The work in this section narrows the focus to research done specifically looking at network threat detection and classification. Many of the methods used to classify network traffic threats are the same as those used for classifying general network traffic so, to avoid unnecessary redundant discussion, they will not be discussed in depth in this paper. They are, however, listed in Table 2.1 so the reader may research them further if so desired. Related work presenting methodologies not already covered in the earlier section will, however be discussed in depth in this section.

Table 2.1: Summary of Previously Covered Methodologies Applied to Network Threat Detection

Author(s)	Year	Methodology	Outcome
Panda and Patra [38]	2007	Naïve Bayes	Overall detection rate: 95%; False positive rate: .02% - 26%
Portnoy et al. [39]	2001	Clustering	Detection rate: 18.56% - 56.25%; False alarm rate: .3% - 11.37%
Zanero and Savaresi [40]	2004	Clustering, Payload Inspection	SOM performed better than PDDP and K-means
Pan et al. [41]	2003	C4.5, Neural Networks	Average detection rate: 85.01% -93.28%; False positive rate: .2% - 19.7%
Moradi and Zulkernine [42]	2004	Neural Networks	Accuracy: 86% - 90%
Xu and Wang [43]	2005	SVM, PCA	Accuracy: 58.3% - 99.9% (class dependent)
Hu W. et al. [44]	2008	AdaBoost	Detection rate: 91.21%; False alarm rate: 3.14%
Linda et al. [45]	2009	Neural Networks	Detection rate: 66.06% - 100%; False alarm rate: 0% - .378%

The Learning Rules for Anomaly Detection (LERAD) algorithm developed by Mahoney and Chan is based on association rule mining [46]. The research takes advantage of the network traffic characteristic of being time series data with long range dependencies. The purpose of the algorithm is to find conditional rules that spot rare events in a time series of tuples, or sequences, of attributes. The long range dependency can be seen as the number of matching attribute values between two tuples lessens inversely to the time interval between the tuples. The two sets of attributes used to test LERAD were IP packets and TCP streams and the dataset was restricted to only the first few inbound packets for each flow. Experimentation with the combined sets of attributes resulted in an average classification accuracy of 50%.

Genetic algorithm based feature selection is used in combination with a decision tree classifier in Stein et al. [47]. The iterative process begins with the random generation of a population where each individual has genes representing the feature set. Each gene receives a value of one or zero depending on whether or not the feature is used in building the decision tree. A decision tree using C4.5 is built for each individual and tested with validation datasets. Fitness of each individual is assessed based on the classification error rate. The genetic algorithm then begins generating the next generation of the population based on those fitness values. The process is repeated with each newly generated population until it reaches a set number of generations. The average classification accuracy ranged from 80% - 99% depending on which category of attack was being considered.

Linda et al. [48] present a fuzzy logic based anomaly detection system. The learning algorithm creates a fuzzy rule base that characterizes previously seen behavior

patterns for standard communication. The rule base is constructed using a real-time version of the nearest neighbor clustering algorithm with the incoming packets; clusters are then converted into the individual fuzzy rules. Real-time processing is made cost effective because the algorithm learns directly from the streaming data and makes storing the packet data unnecessary. The features used were developed in [45] and consisted of window-based statistics generated as a window of specified length was shifted over the stream of packets. Feature vectors are calculated from the packets inside the window as the new packets enter and the last packets exit. The fuzzy rule base is applied to the new input data which is then labeled as anomalous or normal. Experimentation resulted in 71 fuzzy rules created with an average classification of 99.36% and no false positives.

Faloutsos proposes a method of using traffic dispersion graphs for threat detection [49]. The traffic dispersion graphs visualize the communication paths between the hosts and can model interactions such as the type and number of packets. The term “link homophily” is used to represent the tendency of network traffic flows with common IP hosts to share the same application. Link homophily in the network data reveals statistical dependencies between flows with common IP hosts that can be used for traffic classification without requiring information on the packet content or properties. The research introduces a new algorithm called the Neighboring Link Classifier with Relaxation Labeling which requires no training phase or feature generation. The algorithm was used in combination with the traffic dispersion graphs and reportedly worked successfully with botnet detection (a collection of computers used for network attack), however no numerical results were provided.

2.5. Feature Selection Using Neural Networks

In [22] the authors demonstrate how reducing the dimensionality of the features, using FCBF, can improve the technique's classification accuracy. This is shown again in [31], by way of multilayer neural networks. Feature reduction is addressed by inspection of the weights associated with each of the input nodes, eliminating those with the relatively smaller values. From this, the authors reduce the number of features down to 128 for all data sets and 20 "important" features for most data sets. Other examples include [27], using CON and CFS methods, [37] where an embedded Chi-squared is used to determine the value of each feature, and [34] where features are chosen using an MRMR technique based on maximum statistical dependency. While many methods exist for performing feature selection, we will focus on different techniques using neural networks as this leads in to the method chosen for this thesis.

Setiono and Liu look at feature selection using a feedforward neural network with backpropagation [50]. Their methodology begins with all the features and prunes the irrelevant features one at a time. For each feature, the neural network's classification accuracy is calculated with that feature's weights set to zero. The feature that results in the smallest decrease in classification accuracy is removed. This process is repeated until the accuracy rate drops below and decided level. The cross-entropy function, in combination with a penalty function based on the magnitude of each connection's weights, is used as the measurement to minimize during network training. The algorithm was tested with several datasets, both generated and real-world, resulting in a statistically significant improvement in classification accuracy with the selected features versus the accuracy with all of the features.

Belue and Bauer [51] proposed a methodology incorporating a known irrelevant feature (noise) into the set of features. Any feature with a saliency measure falling within a confidence interval around the saliency measure of the noise feature would be removed. Steppe and Bauer [52] built upon this methodology by requiring a paired-t test to account for naturally paired feature saliency observations, and a Bonferroni-type test to demonstrate statistical confidence. Two saliency measures are assessed in this research: derivative-based and weight-based. The iterative process begins with the addition of the noise feature to the feature set followed by generation and training of a set number of neural networks. The saliency measure is calculated for all features and compared to the saliency measure of the noise feature to see if they are statistically different. Features found to be statistically different are kept and the rest are eliminated. The neural network is then retrained using only the retained features. Experimentation with two separate datasets demonstrated an improvement in network performance for both with the reduced feature set.

Two terms are added to the cross-entropy cost function in order to constrain the derivatives of neural network output and hidden node transfer functions in research done by Verikas and Bacauskiene [53]. The network is then trained by minimizing the modified cost function and feature selection is determined by the response of the classification error after features are removed. Once the neural network is generated and trained, the classification accuracy is calculated by setting each feature, one at a time, to zero. The feature removal resulting in the lowest drop in accuracy is eliminated. This continues until only one feature is left. The entire process repeats for each neural network generated and the expected feature rankings and accuracy are calculated by taking the

average results from all of the networks generated. The features deemed salient, based on the set level of accuracy required, are kept and the neural network retrained. Four datasets were used comparing the method proposed in [53] to five other feature selection methods. This experimentation resulted in the proposed method achieving at least slightly higher classification accuracy on all tested data than any of the other tested methods.

The method that compared closest to the one proposed in [53] comes from Bauer et al. [54], and is part of the methodology used in this thesis. The focus of [54] is using the signal-to-noise ratio (SNR) in determining salient feature selection. This research is an extension of the research done in [51] and [52], and proposes the SNR saliency measure which compares the weight-based saliency measure of a feature to the weight-based saliency measure of an injected noise feature. The SNR saliency measures for irrelevant features should be less than or close to zero while the measures for salient features should be significantly larger than zero. The higher the SNR saliency measure, the higher the saliency of the feature. The SNR method was applied using three different datasets and compared against the performance of the algorithm developed in [50] and a method using Principal Component Analysis (PCA). The SNR method performed comparably against the other two methods while only requiring one feature versus an average of 2.7 features for [50] and nine features for the PCA method.

2.6. Summary

This chapter presented highlights of work related to the work done in this thesis. The chapter opened with background information to give the reader some familiarity with computer networking terms and concepts. Next, the broad category of general network traffic classification was explored, looking at port and signature-based analysis as well as

different aspects of unsupervised and supervised machine learning algorithms including clustering and neural networks. The focus then narrowed to threat detection classification examining methods not discussed in the general traffic classification section. Narrowing further, the chapter concluded by focusing on previous work done on salient feature selection using neural networks and the SNR saliency measure.

III. Methodology

3.1. Chapter Overview

This chapter describes the methodology used in this thesis. First is a description of the dataset, including the quantity and type of data, and the data collection process. Next is a breakdown of the overall methodology. The discussion continues with an explanation of the data preprocessing, a major endeavor, using a feature generator with some challenging requirements and specialized network security software to create ground truth data. The chapter then provides a brief summary of the science behind neural networks and salient feature selection, followed by a description of the software-based tools necessary for the data analysis. The chapter concludes with a discussion of the performance metrics used to evaluate the performance of the neural network classifier.

The focus of this research is the processing and analysis of network traffic capture data to determine the salient features when threat detection is of primary interest. Massive amounts of traffic data can pass through a network quickly and the sheer magnitude of the data makes analysis both difficult and untimely. If those particular features of the traffic data that provide valuable threat-assessing information could be determined, they could be focused on, reducing the necessary processing and analysis time, while enabling and enhancing network security.

3.2. Dataset Description

Data for this research comes from the Cyber Defense Exercise (CDX) sponsored by the US National Security Agency (NSA). A description of the data and the situation/environment it was collected in can be found in Mullins et al. [55]. CDX is an annual competition held between the US Military Academy at West Point, US Air Force

Academy, Naval Postgraduate School (NPS), US Naval Academy, US Coast Guard Academy, US Merchant Marine Academy, and Air Force Institute of Technology (AFIT). The exercise provides military students the opportunity to apply defensive information assurance best practices in a real-world-modeled environment. During the exercise, the NSA, along with highly trained operators from the services' network operations centers, acts as a "Red Team" of hackers, launching cyber attacks on the networks designed and defended by the students.

3.2.1. Data Collection

As an exercise based on real-world situations, the CDX data was deemed an acceptable representation of the type of traffic a military network might face and thus, a good choice to use for this research. The network traffic data used was collected during the CDX from 2003 – 2007, and 2009. Having several years worth of data frees the analysis from being restricted to the year the data was collected (as techniques and exploits are constantly evolving and adapting).

3.2.2. Collection Equipment

AFIT's part of the CDX took place in the Laboratory for Information System Security/Assurance Research and Development (LISSARD), a subsection of the school's Graduate Education Cyberspace Operations (GECO) laboratory. Equipment used for the CDX varied from year to year but consisted primarily of Dell and Cisco brand information technology (IT) products [55]. Traffic was collected off the firewall, a Dell server with two network ports, and captured on the external port to get all traffic coming into and going out of the network. The TCPDump software, running on an OpenBSD Linux distribution, handled the actual traffic collection.

3.2.3. Dataset Type and Size

The network traffic collected during the CDX was captured in the libpcap file format (commonly denoted as pcap) which is the standard format for network capture tools such as TCPDump [56], used mainly with Linux-based operating systems, and Wireshark [57], used frequently in Microsoft Windows operating systems. The pcap files consist of the packets transmitted between the hosts and clients for that specific collection time period. Figure 3.1 is a screenshot of an example packet captured in Wireshark. Information in the packet includes (from top to bottom) Frame, Ethernet, IP, and TCP parameters. The parameters listed will depend on the type of protocol the packet is sent as (e.g., TCP, UDP or ICMP). We can see the packet from Figure 3.1 was sent using TCP because it includes the TCP parameters. The very bottom of the screenshot shows the contents of the packet's payload.

Table 3.1 shows the total number of packets captured for each year of the provided dataset. The total number of packets in the full dataset is 12,145,569. Rather than focus at the packet level, the data was separated into flows designated by the four-tuple of source and destination IP addresses and port numbers. This occurred during the data preprocessing stage. Typically, flows are created by the five-tuple which includes the previously mentioned four-tuple and the internet protocol type. Most of the packets' internet protocols in the CDX dataset were TCP; however, in order to eliminate any protocol-based restrictions on the results, no specific efforts were made to remove other protocol types and the four-tuple was used instead. Traffic was considered in all directions: client-to-server, server-to-client, and back and forth between client and server.

Figure 3.1: Screenshot of an Example Network Packet

The traffic captured also had no requirement to get a complete traffic flow, meaning the flow captured does not have to include the SYN and final (FIN) packets. Removing this limitation allows for interpretation even if the traffic capture occurred mid-flow or does not continue until transmission is complete. Merging the packets into

flows reduced the total number of observations to 2,048,918 (a flow can consist of one or more packets). The yearly flow breakdown is also included in Table 3.1.

Table 3.1: Breakdown of Dataset Packets and Flows

	2003	2004	2005	2006	2007	2009	Total
Packets	1555220	2855596	1156398	1060816	3878980	1638559	12145569
Flows	268884	610601	38821	194001	627327	309284	2048918

3.3. Overall Methodology

The overall methodology for this research is broken down into two parts: data preprocessing and neural network analysis. Each part of the methodology is explained in more detail in the sections that follow. A flow chart is shown in Figure 3.2 to provide a visual representation of the overall methodology and how its parts fit together.

3.4. Data Preprocessing

A large portion of the work for this research dealt with preprocessing the data for analysis. Converting the provided dataset, containing packet information like in Figure 3.1, into data readable by neural networks was both challenging and time consuming. The preprocessing began with feature extraction to create the flows or observations. The threat labels then were created and matched up with their corresponding observations. From there, the data went through a cleanup process involving the removal of observations containing incomplete data and the removal of information-less features. Finally, the data was randomly separated into balanced sets, containing equal numbers of observations per class. Other, more current, software solutions may be available to generate the attributes without this level of preprocessing, but were unavailable at time of writing.

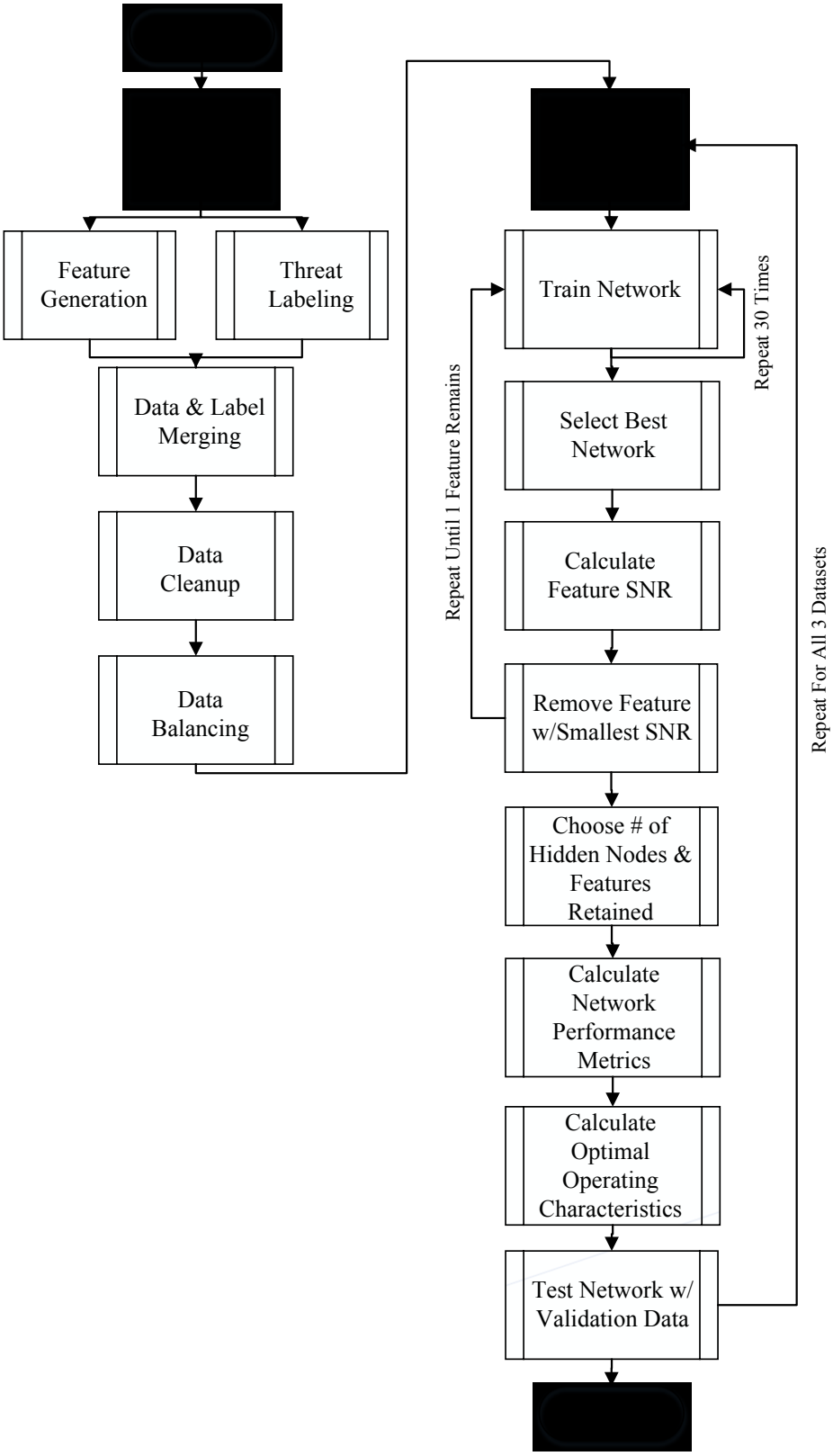

Figure 3.2: Overall Methodology Flow Chart

3.4.1. Feature Generation

Feature generation from the data was made possible by the fullstats.v1.0 (Fullstats) Perl script created by Moore [22]. It is available for download from the University of Cambridge Computer Laboratory Downloads: BRASIL – Characterizing Network-based Applications page [58]. A description of the list of features generated from the packet capture data is in Appendix B: Original Feature List.

3.4.1.1. Software Requirements

Fullstats consists of three scripts that call on functionality from previously installed software packages. The three scripts include a flow creator, an attribute generator, and a script to convert the attribute output into different file formats. Because Fullstats was created several years ago, it requires functionality no longer supported in current versions of the necessary installed software packages. Information on the specific versions of the software packages required to run the script was provided by Ji [59]. The Fullstats script and its required software packages ran on the Ubuntu (Linux-based) OS version 5.10, referred to as "Breezy Badger", set up as a virtual machine using Oracle Virtual Box. The software package versions used in this research include, GCC 4.0.1-3, Perl 5.8.7-5, TCPDump 3.9.1-1, TCPTrace 6.6.1, TCPDemux 20050725, and TCPSlice 1.2a3. Most of the software listed here had to be retrieved from their individual archives or SourceForge.net as the update repositories have long been closed. Figure 3.3 shows a screenshot of the Fullstats attribute generator running in the virtual machine. Displayed there, from left to right, is the file number of the current file being processed, processing bit rate, processing frame rate, current file storage location, and estimated time of overall processing completion.

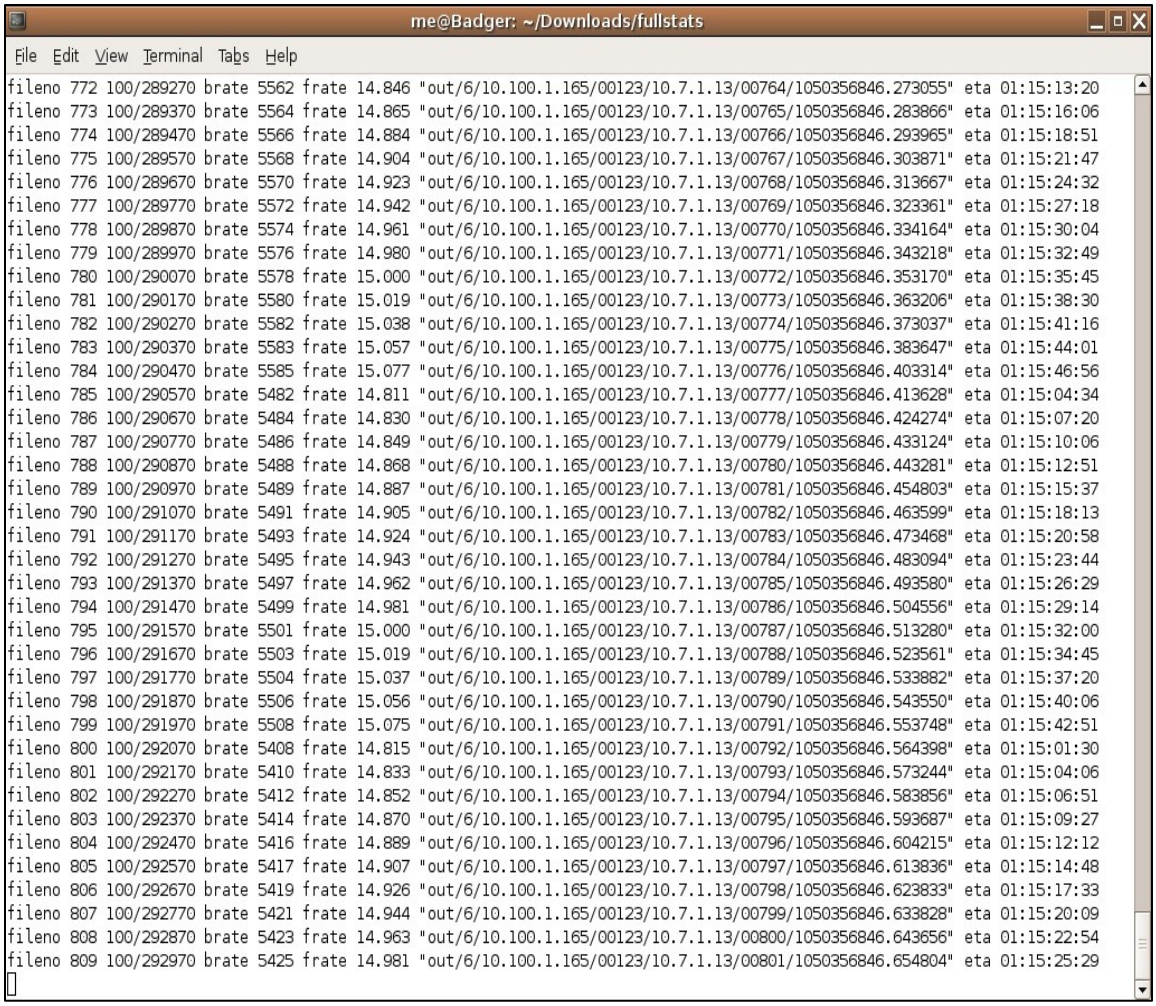

Figure 3.3: Fullstats Attribute Generator

3.4.1.2. Equipment

Due to the size of the provided dataset, and the time required to run all of it through Fullstats, processing required multiple computers. Five dedicated Dell PCs in AFIT's GECO Laboratory were used, all running Microsoft Windows 7 Enterprise edition with Service Pack 1. Each PC had Intel Xeon 3GHz multi-core processors ranging from 6 to 8 cores and 20 to 32 GB of RAM. It took the five computers about 600 hours total to process all of the data through Fullstats. Figure 3.4 is an image of the GECO Laboratory where the data was processed.

Figure 3.4: AFIT GECO Laboratory

3.4.1.3. Processing Issues

A few issues came up during the Fullstats feature extraction that needed to be addressed to allow the script to work successfully. First off, the script for converting the attribute output to a particular file format had to be modified to include an "unknown" traffic direction, possibly because the original code was designed to focus on just the TCP protocol. This research makes no distinction between the TCP and UDP protocols, considering flows using both formats. Secondly, the large size of the capture files frequently caused the script to abruptly halt, so TCPDump was used to split the large capture files into capture files small enough for Fullstats to process. Finally, about 90% of the data was rendered useless because Fullstats was unable to completely process

many of the manually created and/or truncated packets found in the network traffic generated during the exercise. The sheer magnitude of the data, however, made this less of a concern, as there were still over 204,000 complete observations, across all years captured, available for analysis.

3.4.2. Threat Label Creation

One of the most difficult issues when attempting to analyze network traffic is acquiring the ground truth data. Truth data was not provided for the CDX, during or after the exercise; team points were assessed based on how well the targeted systems were kept operational, not on the specific attacks or defensive techniques [55]. Because of the lack of truth data, another method of labeling the observation classes was developed using a network security-based Linux OS and an intrusion detection engine.

3.4.2.1. Equipment

Processing from this point forward took place on two computers: a custom-built PC with an AMD Phenom II 3.2GHz 4-core processor with 8GB RAM and a Hewlett-Packard Envy model laptop with an Intel Core i7 4-core processor with 8GB of RAM. Both computers were running Windows 7 Home Premium edition with Service Pack 1.

3.4.2.2. Security Onion

Security Onion is a network security-based OS developed by Doug Burks [60]. The OS is based on the Ubuntu version of Linux and includes a plethora of network security monitoring, intrusion detection, and log management software. The version of Security Onion chosen was 12.04.3-20130904 64-bit, running in an Oracle Virtual Box virtual machine. The software tools used included TCPReplay [61], to play back the

capture files through the intrusion detection system sensor, and Sguil [62], a graphical user interface (GUI) console designed for use with the Snort intrusion detection engine.

3.4.2.3. Snort Intrusion Detection System

Snort is signature-based intrusion detection system created by Sourcefire, now a branch of Cisco [63]. Intrusion detection is handled through the use of customizable rule sets which decide how the traffic should be handled based on what matches up with packet header or content information. The Security Onion OS came with Snort version 2.9.5.3. The rule set used for this research was created and released by developers at Snort.org and was current as of November 7, 2013. Because the most recent capture data came from the exercise in 2009, it is assumed the Snort rule set included any threat it would encounter with this data set.

3.4.2.4. Sguil

The Sguil console provides a human-interpretable representation of the threats discovered by the Snort intrusion detection system. The threat levels, assessed by the rule sets in the Snort engine, are differentiated by different colors. The Security Onion OS included Sguil version 0.8.0. The virtual machine's network adapter was set to internal network only to keep any outside traffic from interfering. A MySQL-based query capability allows the user to search for and export labeled threat data. In the sequential label creation process, the capture files were run through the intrusion detection system using TCPReplay (version 3.4.3) with the Sguil interface opened to display the threat assessment output. A query using the TCPReplay start time provided a listing of the capture files' threat-labeled flows which was then exported into a comma separated value (CSV) file format. The exported file also included the flows' source and destination IP

addresses and port numbers which was used later on to merge with the observations created by Fullstats.

Figure 3.5 shows a screenshot of the Sguil GUI. In the screenshot, we can see the assessed threat levels on the left side of the side in yellow, orange, and red. Corresponding parameter information makes up the rest of the table. On the bottom right we can see the parameters and payload of the highlighted listing.

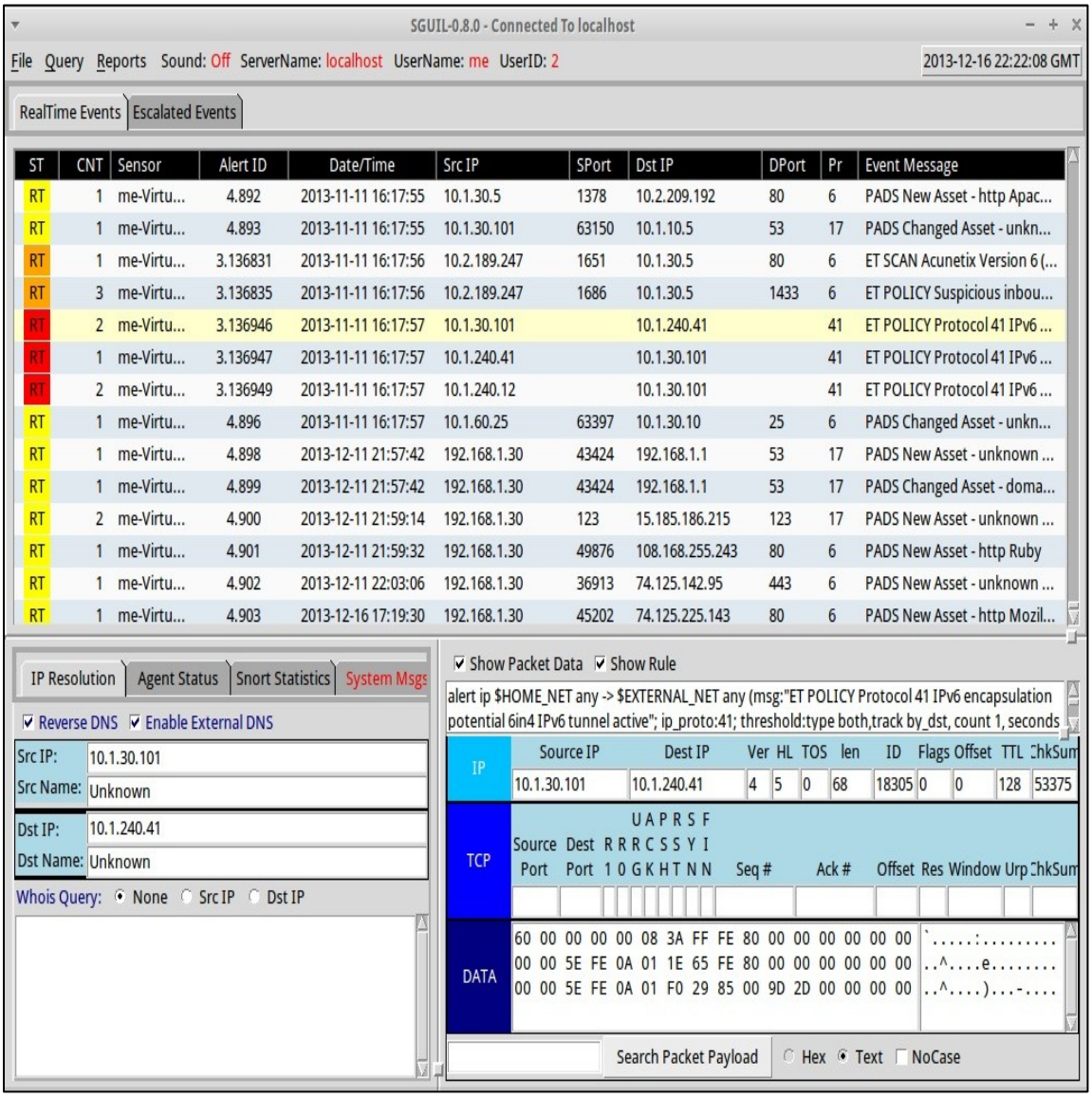

Figure 3.5: Sguil GUI Screenshot

3.4.3. Observation Threat Labeling

The primary output of Fullstats is the ARFF file format, intended for use with the WEKA Java-based data mining software developed at the University of Waikato in New Zealand [64]. This was an issue because the label files intended to merge with the observation data were in the CSV file format. While the WEKA software has many analysis tools included in it, analysis for this research was to be done using the neural net tool in MATLAB, which cannot read in ARFF files without additional added functionality. Conversion from ARFF to CSV is possible; however, Fullstats is also capable of outputting CSV file format versions of the observation data, making conversion unnecessary. Another benefit of the CSV output files from Fullstats is the inclusion of the IP and port information for the flows. This information was imperative in matching up the labels with their corresponding observations.

The threat levels provided by Sguil consist of levels 1-5, with 1 being the highest threat, and 5, the lowest. For simplification, threat labels in the label files were converted to a 1 if the Sguil level was 4 or 5, 2 if the Sguil level was 2 or 3, and 3 if the Sguil level was 1, allowing for the no-threat level to be represented by a 0 and escalating from there. A macro written in Visual Basic for Applications (VBA) in Microsoft Excel allowed for an automated method of merging the observation and threat label data, based on the previously mentioned four-tuple of source and destination IP addresses and port numbers. This macro provided a large time-savings considering the number of observations in the data set.

3.4.4. Data Cleanup

Three steps were involved in cleaning up the data and preparing it for analysis. The first step dealt with missing data. As mentioned earlier, approximately 90% of the packet capture data was unable to be completely processed by Fullstats; this created many observations with missing information. Once the observation files were merged with the label files, the observations with missing data needed to be removed. Another macro was written in Excel to automate the searching and removal process.

Some of the observation data contained the letter Y or N in response to whether or not the flows met certain criteria (e.g., window scaling factor was used). To keep the data numerical the Ys were changed to 1s and the Ns were changed to 0s in the second step of data cleanup.

The third step involved the removal of features providing no valuable information to the analysis. Features specific to the requirement of having a complete flow (capturing the SYN and FIN packets from the flow) were removed because the experiment was specifically designed to include incomplete flows (allows for broader interpretations). Other features were determined as having zero variance. If all entries in the feature are the same then the feature provides no new information to aid in classification. After the datasets were balanced (see section 3.3.5), a couple simple lines of MATLAB code computed the standard deviation of the provided data set and removed the indicated features. Removing the information-less features prior to analysis helps the neural net tool function correctly. Finally, the features for server and client port numbers were removed because port number analysis has been shown to be a poor predictor [16].

3.4.5. Data Balancing

Of the 204,000 observations, only about 16,000, or 8% of the data, were labeled as a threat, making the data extremely unbalanced. To give the neural net tool the best chance of success, new, smaller data sets were created. The pseudo-random number generator in Excel was used to randomly select an equal number from each class for each desired data set. Using this method was deemed acceptable because the random number function in versions of Excel 2003 or later pass the standard tests of randomness referred to as Diehard [65]. Three separate datasets were developed. The first, and largest set, is intended to determine whether or not traffic should be considered a threat (all threat level data combined into one class). The second set is designed to determine how well the different threat levels could be distinguished. The third set is a "complete" data set, meant to determine whether or not a threat was present and if so, the level of that threat. The new data sets were coded as *Threat vs. No-Threat* (No-Threat, Threat), *Threats Only* (Low, Medium, High), and *Complete* (None, Low, Medium, High). After the new data sets were created, 10% of each class in each data set was withheld as validation data.

3.4.6. Final Dataset Description

The number of alert observations versus the number of overall observations broken down by year is shown in Table 3.2. The small number of observations with threat level 1, the lowest level, greatly reduced the overall number of observations for the *Threats Only* and *Complete* datasets. Table 3.3 provides the breakdown of the final data sets chosen for analysis, including the number withheld for validation. There are an equal number of observations of each class in each data set. The table also includes the number

of features each dataset begins with after data cleanup is complete (not including the label feature).

Table 3.2: Yearly Breakdown of Full Dataset

	2003	2004	2005	2006	2007	2009	Total
Observations	30829	73507	1301	9535	51777	37422	204371
Threat observations	3357	8628	39	214	41	4037	16316
Percentage	10.89%	11.74%	3.00%	2.24%	0.08%	10.79%	7.98%

Table 3.3: Final Datasets for Analysis

	Threat vs. No-Threat	Threats Only	Complete
Observations	29369	516	688
Withheld	3263	57	76
Total	32632	573	764
Starting Features	229	222	224
Classes	2	3	4

3.5. Neural Network Analysis Methodology

The main focus of this research is to determine which features, derived from network traffic, are the most important to determining if a threat is present on the network. The methodology chosen to accomplish this is the feed-forward artificial neural network using backpropagation using signal-to-noise ratio for feature selection. This section discusses both the neural network concepts and the tools this research uses to apply the neural network classification capabilities to analyze the final datasets.

3.5.1. Neural Networks

Neural networks are a method of supervised machine learning modeled by the learning abilities of biological cognitive systems (i.e., neurons in the brain) [66]. The neurons are networked together to allow communication and information processing. The learning takes place through feedback causing parameter adjustments intended to make

the output more accurate. Figure 3.6 illustrates an example of a fully connected multilayer perceptron (MLP) artificial neural network.

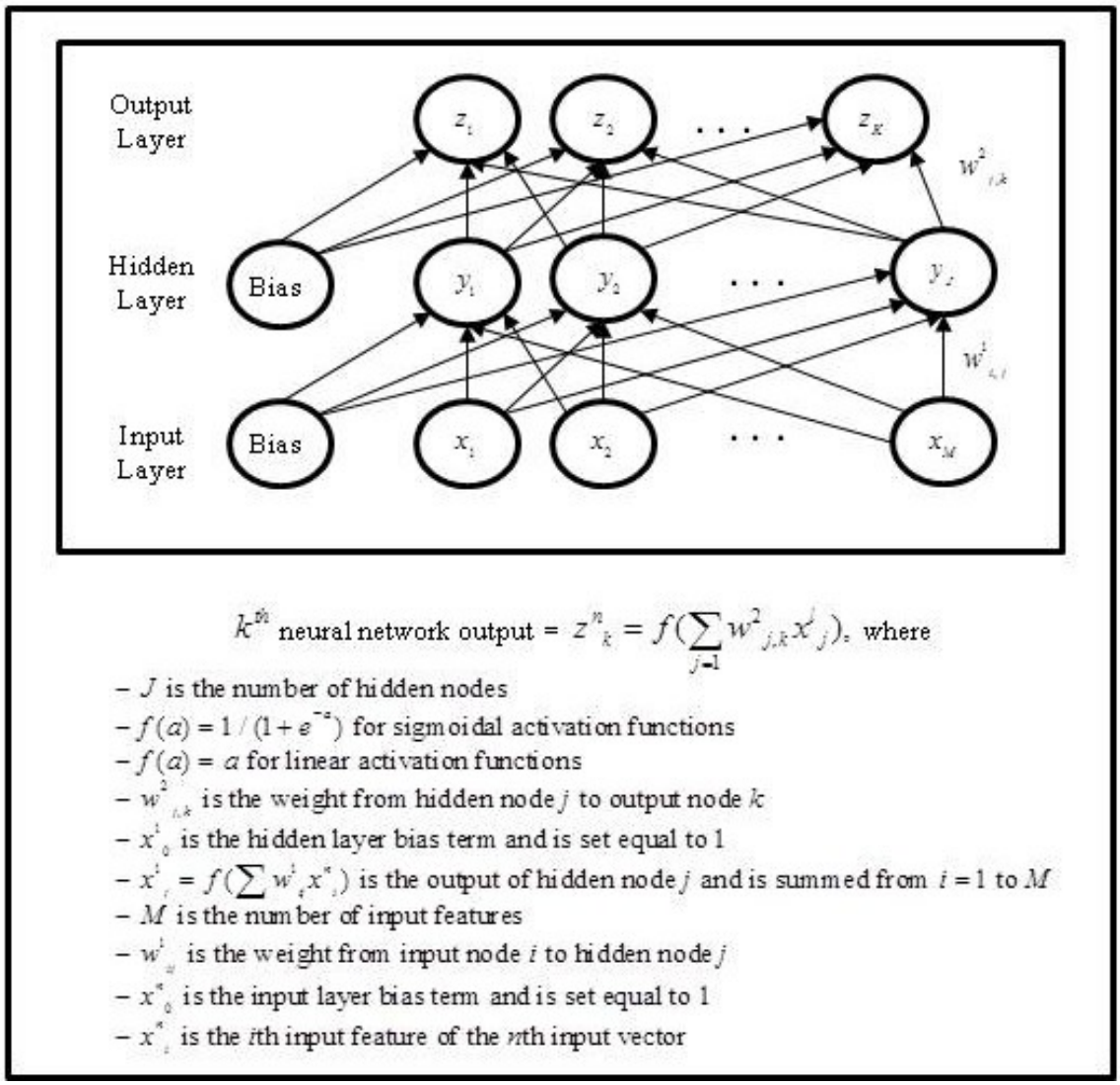

Figure 3.6: Fully Connected MLP ANN Example [66]

A weighted combination of the inputs is created and the data is transformed through a threshold logic or transformation function. Examples of transformation functions include hard limiting, hyperbolic tangents, and sigmoid functions. This research uses the sigmoid function as the transformation function for the neural networks to address because it addresses the non-linearity introduced by the hidden layers, is

continuous (differentiable), and has a limited range (0 to 1) but never reaches a maximum or minimum [67]. The next few sections present definitions of terms used with neural networks, as well as the specific algorithm and saliency measure used in this research.

3.5.1.1. Definitions

The following are definitions of terminology used when describing the neural net methodology. The definitions come from the class notes used in OPER685 Multivariate Analysis I, Spring 2013 [66].

- *Activation Function*: defines the output of a node given an input or set of inputs
- *Artificial Neural Network (ANN):* an information processing system (algorithm) that operates on inputs to extract information and produces outputs corresponding to the extracted information
- *Architecture:* the topological arrangement of neurons, layers, and connections, which defines the set of modeling equations available to the ANN
- *Backpropagation:* a learning algorithm for updating weights in a feed-forward MLP ANN that minimizes the (e.g., mean squared) mapping error
- *Epoch:* a complete presentation of the dataset being used to train the MLP, or equivalently called a training cycle
- *Feature:* in neural networks, features refer to the input vectors of information which are presumed to have some relation that may be helpful in distinguishing the various output classes; vector of features is often called an observation
- *Feed-forward*: multilayer ANNs whose connections exclusively feed inputs from lower to higher levels; in contrast to a feedback or recurrent ANN, feed-forward ANNs operate only until all the inputs propagate to the output layer

- *Hidden Units*: processing elements in MLP ANN that are not included in the input or output layers; the part of the neural network located between the input and output layers
- *Learning Algorithm*: equations used to modify the weights of processing elements in response to input and output values
- *Neuron*: fundamental building block of an ANN; normally, each neuron takes a weighted sum of its input to determine its net input which is then processed through a transfer function to produce a single-valued output that is broadcast to 'downstream' neurons
- *Perceptron*: a type of ANN algorithm used in pattern classification problems that is trained using "supervision"; can be single or multilayer; connection weights and thresholds can be fixed or adapted using a number of different algorithms
- *Supervised Training:* a method of training adaptive ANNs that requires a labeled training dataset and an external teacher; using the desired response, the teacher provides responses for correct of incorrect classification by the network
- *Weight:* processing elements (or neurons or units) receive inputs by means of interconnects (also called 'connections' or 'links'), each of which has an associated weight, signifying its strength; the weights are combined to calculate the activation functions

3.5.1.2. Algorithm

The algorithm chosen for this research is the Instantaneous Backpropagation Algorithm for a Single Hidden Layer Feed-Forward Neural Network; its steps are as follows from [66, 68]:

1. Randomly partition data into *training, training-test*, and *validation* sets.
2. Normalize the feature input data.
3. Initialize weights to small random values.
4. Present the network with a randomly selected vector from the training set, denoted x^p.
5. Calculate the network output z^p associated with the p^{th} training vector.

   K^{th} neural network output: $z_k^p = f\left(\sum_{j=0}^{H} w_{jk}^2 x_j^1\right)$, where
   - H is the number of middle nodes
   - $f(a) = \frac{1}{(1+e^{-a})}$ for sigmoidal activation functions
   - $f(a)$ for linear activation functions
   - w_{jk}^2 is the weight from middle node j to output node k
   - x_0^1 is the middle layer bias term and is set equal to 1
   - $x_j^1 = f\left(\sum_{i=0}^{M} w_{ij}^1 x_i^p\right)$ is the output of middle node j
   - M is the number of feature inputs
   - w_{ij}^1 is the weight from input node i to middle node j
   - x_0^p is the input layer bias term, and is equal to 1
   - x_i^p is the i^{th} feature input
6. Update the weights.
   - Upper layer weights: $(w_{jk}^2)^+ = (w_{jk}^2)^- + \eta\delta_k^2 x_j^1$,
   - Lower layer weights: $(w_{ij}^1)^+ = (w_{ij}^1)^- + \eta\delta_j^1 x_i^p$, where
     - $(w_{jk}^2)^+$ is the updated weight from middle node j to output k
     - $(w_{jk}^2)^-$ is the old weight from middle node j to output k
     - $(w_{ij}^1)^+$ is the updated weight from input i to middle node j
     - $(w_{ij}^1)^-$ is the old weight from input i to middle node j
     - η is the step size

- $\delta_k^2 = (d_k^p - z_k^p) z_k^p (1 - z_k^p)$, if there is a sigmoid on the output
- $\delta_k^2 = (d_k^p - z_k^p)$, if the output is linear
- $\delta_j^1 = x_j^1 (1 - x_j^1) \sum_{k=1}^{K} \delta_k^2 (w_{jk}^2)^-$, if there is a sigmoid on the middle node *j*
- $\sum_{k=1}^{K} \delta_k^2 (w_{jk}^2)^-$, if middle node *j* is linear
- d_k^p is the k^{th} desired output of the p^{th} exemplar

7. If training-test set error does not indicate sufficient convergence, go to step 4.

3.5.1.3. Saliency Measure

The saliency measure is used to determine feature relevance in order to find a parsimonious feature set. The focus of this research is to determine which features of the dataset are salient. Two types of salient measures for neural network feature selection are derivative-based and weight-based [69]. The measure chosen for this research is the weight-based signal-to-noise ratio (SNR) saliency measure. This was discussed in Bauer et al. [54]. A simple weight-based saliency measure is computed as, shown in [70]:

$$\tau_i = \sum_{j=1}^{J} (w_{i,j}^1)^2 \text{, where}$$

- τ_1 is the measure for feature i, J is the number of hidden nodes, $w_{i,j}^{\ 1}$ is the first layer weight between input node i and hidden node *j*
- The measure is simply the sum of the squared weights between input node i and all hidden nodes 1 through J

The SNR measure directly compares the saliency of a feature to an injected noise feature. The measure expands upon the simple weight-based computation as:

$$SNR_i = 10\log_{base10}\left(\frac{\sum_{j=1}^{J}(w_{i,j}^{1})^2}{\sum_{j=1}^{J}(w_{N,j}^{1})^2}\right), \text{ where}$$

- SNR_i is the value of the saliency measure for feature i, J is the number of hidden nodes, $w_{i,j}^{1}$ is the weight from node i to node j, and $w_{N,j}^{1}$ is the first layer weight from the noise node N to node j
- The injected noise is created as a Uniform (0,1) distribution
- The scaled logarithmic transformation of the ratio converts the saliency measure to a decibel scale

The idea behind the SNR saliency measure is that if a feature is relevant to the output, its weights will be higher, thus making the SNR higher [54]. The noise feature is added to the set of features, the features are standardized to zero mean with unit variance, the weights are randomly initialized and assigned, the neural network is generated, and the SNR for each feature is calculated. The feature with the lowest of the calculated SNR values is dropped, the neural network generation begins again, and the process is repeated until only one feature and the noise feature remain

3.5.2. MATLAB Neural Network Tool

Due to the complexity of the neural network and saliency measure calculations, in combination with size of the dataset, a software tool is required for data analysis. The primary tool used to analyze the final datasets is the Mathworks MATLAB Neural Network toolbox. From the website, "Neural Network Toolbox™ provides functions and apps for modeling complex nonlinear systems that are not easily modeled with a closed-form equation. Neural Network Toolbox supports supervised learning with feedforward,

radial basis, and dynamic networks. With the toolbox you can design, train, visualize, and simulate neural networks. You can use Neural Network Toolbox for applications such as data fitting, pattern recognition, clustering, time-series prediction, and dynamic system modeling and control." [71].

3.5.2.1. Tool Specifics

For standard pattern recognition without encompassing feature reduction/selection, one tool available, from the Neural Network Toolbox, is the pattern recognition tool or "nprtool". Observation data is loaded into nprtool as inputs and observation labels (truth data) are loaded as outputs. Calling the tool from the MATLAB command line opens a GUI for the user to select the parameters. The GUI walks the user through selection of the input data, the target or output data, how the data should be broken into training, testing, validation sets, and the number of hidden neurons in the middle layers. From there, the user selects "train" to train the neural network on the data provided, which can be repeated until satisfactory results are achieved. Next, the network is evaluated and a number of deployment options are provided, finishing with the ability to save the generated network.

3.5.2.2. Code Modifications

The nprtool GUI is adequate when looking at smaller feature sets, as analysis occurs at a one-at-a-time rate, but very cumbersome when the feature sets are large. A nice feature of the GUI is that it allows a script to be generated based on the steps taken using the GUI. This script can then be used as the base code and modified to include the desired parameters and allow for automation. When the modified code for this research is used the nprtool GUI does not show; instead, the neural network training GUI appears,

allowing the user to view the network generation process if so desired. The rest of the processing takes place in the background. Figure 3.7 provides a screenshot of the neural network training GUI. For this research, the code was modified to include automation of the neural network processing through user-defined network structure, noise feature creation, "best" network selection, tracking of the features dropped, and a tailored data storage system to make sure all data is captured throughout the processing. The modified code can be found in Appendix C: MATLAB Code for Neural Network Processing.

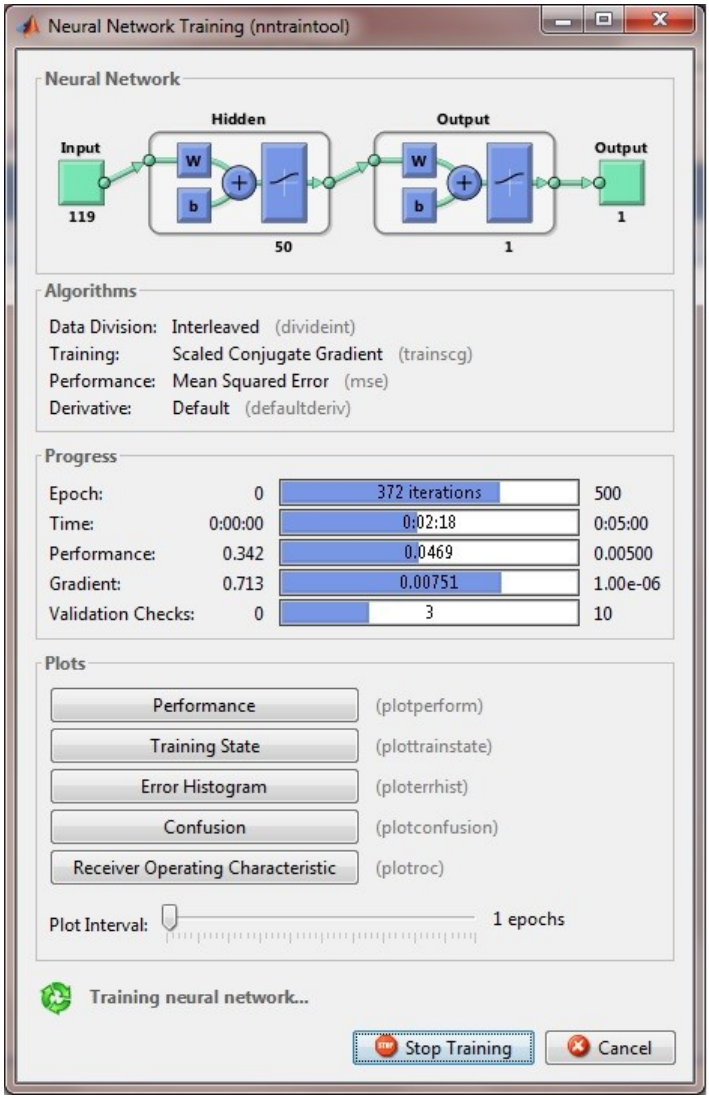

Figure 3.7: MATLAB Neural Network Training GUI

3.5.2.2.1. Hidden Nodes

Neural networks usually consist of three layers, input, middle, and output. The structures of the input and output layers are decided by the number of features in the dataset (input) and the number of classes of the data (output). The structure of the middle, or hidden, layer can play a large part in how well the neural network performs. Weights are applied to the data as it passes from the input layer to the hidden layer and again, as the data passes from the hidden layer to the output layer. The number of neurons in the middle layer, called hidden nodes, determines the complexity of the network. More nodes may generate a better performing network, but it also increase network complexity and processing time. The code was modified so the user can choose the number of hidden nodes in the network offering the ability to compare the network performance versus processing time trade-off. The number of hidden nodes chosen is explored in Chapter IV.

3.5.2.2.2. Noise Creation

The noise feature is generated from a uniform (0, 1) distribution. The MATLAB "rand" function is used to create the noise feature making it the same length of the dataset (i.e., same number of observations). That noise feature is then appended to the dataset as the first column.

3.5.2.2.3. Net Selection

Another benefit of the modified code is the ability to set the number of attempts made to generate the neural networks. This is a valuable tool because the networks generated occasionally get stuck at a local minimum, causing classification accuracy to be lower than it should. Running multiple attempts of the neural network generation allows a better performing network to be chosen. There are several options available to

use for network performance criteria such as mean square error, sum square error, and cross-entropy. When neural network analysis begins, the performance value is calculated for the network generated with each attempt. The networks are reinitialized at the start of each attempt. The attempt with the best performance value (usually the minimum) is chosen as the best network and the saliency measurements are calculated for the features. The feature with the smallest SNR value is then removed from the feature set and the process begins again.

3.5.2.2.4. Bookkeeping

A necessary aspect of the automated feature removal process is keeping track of what features remain after each removal in order to relate the new list of features with the original feature set. When the feature is removed, the indexes of the features after the removed feature will change. Careful bookkeeping keeps that original structure intact. Separate arrays are used to keep track of both the features remaining (based on the original structure) and those features removed.

3.5.2.2.5. Generated Data Storage

Running the neural network tool generates a large amount of data, from network performance graphs, to confusion matrices, to arrays tracking classification accuracies and feature removal. The modified code creates both cell arrays capable of holding large amounts of multidimensional data and individual plots, tailored for specific purposes, to make sure no valuable information is lost.

3.5.2.2.6. Time Keeping

With such large amounts of data, processing time is of concern. For example, the previously mentioned performance versus processing time associated with the network

hidden layer structure. Another example is the number of attempts made generating the networks. The increase in performance needs to be worth the extra time it takes to process the data. With that in mind, the code was modified to include a timekeeper, or "stopwatch", function. A timer starts when the neural network tool begins and ends with the completion of the last attempt at network generation. The time information is saved and made available for comparison.

3.5.3. Performance Metrics

Evaluating the performance of a classifier is a complicated and focus-specific task. There are several different areas of interest that can define performance such as cost, speed, and accuracy [72]. The focus areas chosen depend on what is defined as important in the classification outcome. The performance metrics used in this research are based on information found in the confusion matrix and include the overall success rate, marginal rates, means measures, and receiver operating characteristic (ROC) curves.

3.5.3.1. Confusion Matrix

A confusion matrix describes how the observation classifications are distributed over the actual and predicted classifications in a grid-like format. The rows represent predicted classes and columns represent actual classes. Table 3.4 provides an example of a confusion matrix for a two class classifier [66]. A confusion matrix contains four values used to derive other performance measures:

- *True Positive (TP)*: percentage of correct positive class predictions; hits
- *True Negative (TN)*: percentage of correct negative class predictions; correct rejections

- *False Positive (FP)*: percentage of incorrect positive class predictions; false alarms; type I errors
- *False Negative (FN)*: percentage of incorrect negative class predictions; misses; type II error
- *Positive (P)***:** number of positive labeled observations
- *Negative (N):* number of negative labeled observations

Table 3.4: Two-Class Confusion Matrix

		Predicted	
		Target	**Noise**
Actual	**Target**	TP	FN
	Noise	FP	TN

3.5.3.2. Overall Success Rate

The most commonly seen classification performance metric is the overall success rate, or percent correct over all instances; also referred to as overall accuracy. Overall accuracy is the trace of the confusion matrix, divided by the total number of instances and ranges from 0 to 1, or perfect misclassification to perfect classification [72]. Accuracy by class is also included. Class accuracy is the percent correct out of each class.

$$Accuracy = \frac{(TP + TN)}{(P + N)}$$

3.5.3.3. Marginal Rates

Classification accuracy is more than just the percentage of correctly classified observations [72]. The marginal rates (i.e., margins of the confusion matrix) provide useful performance metrics as well. The marginal rates metrics used in this research are *recall* (or true positive rate (TPR)), *specificity* (or true negative rate), *precision* (or positive predictive value), *false positive rate* (*FPR* - type I error), and *false negative rate*

(*FNR* – type II error) [73]. Recall measures the ability of the model to correctly predict observations are in a particular class [74]. Specificity is similar to recall, but for correctly predicting observations are not in a particular class. Precision measures the accuracy of a specific class being predicted. All five measures range from 0 to 1.

$$Recall = \frac{TP}{P} = \frac{TP}{(TP + FN)} \qquad Specificity = \frac{TN}{N} = \frac{TN}{(FP + TN)}$$

$$Precision = \frac{TP}{(TP + FP)} \qquad FPR = \frac{FP}{(FP + TN)} \qquad FNR = \frac{FN}{(TP + FN)}$$

3.5.3.4. Means Measures

Two types of means measures are looked at in this research. Means measures focus on the per-class performance. The first is the geometric mean or G-measure. The G-measure is the square root of the product of *precision* and recall. The measure normalizes the true positive to the geometric mean of the predicted and actual positives [73].

$$G = \sqrt{Precision * Recall}$$

The F-measure (or F1 score) is another way to measure a classifier's accuracy. The F-measure corresponds to the harmonic mean of *recall* and *precision*. It measures the overlapping of the actual and predicted classes and ranges from 0 to 1, or no overlap (worst) to complete overlap (best).

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

3.5.3.5. ROC Curves

ROC Curves got their start as a way of explaining radio signals during World War II and have become a commonly used tool in machine learning research communities in

recent years [75]. ROC Curves provide a graphical way to visualize the tradeoff between TPR and FPR based as a function of some varied parameter of the classifier [66]. For this research the parameter is the decision threshold value for deciding which class an observation belongs to. Perfect classification is represented as the point (0, 1) along the curve (see Figure 3.8). The optimal operating point of the curve is the threshold providing the best combination of TPR and FPR.

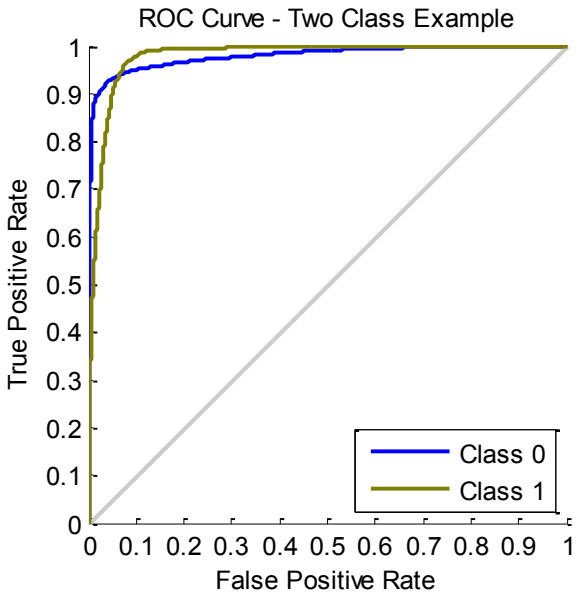

Figure 3.8: Two-Class ROC Curve Example

The area under the ROC Curve (AUC) provides a metric of how well the classifier can predict an observation's class. The AUC is the probability that the classifier will classify a randomly chosen positive instance higher than a randomly chosen negative instance [75]. As with the previous metrics, AUC ranges between 0 and 1, with 0 giving the worst predictive capability and 1 giving the best.

3.6. Summary

The methodology described in this chapter is designed to provide an accurate determination of the most important features in network traffic data for classifying threats. The initial dataset was discussed, followed by the process it took to turn that dataset into something useable for analysis by neural networks. After that, neural networks were explained. The chapter concluded with a discussion of the software tools and modifications necessary for analysis and a description of the metrics used for evaluating the neural network classification performance.

IV. Experimental Analysis & Results

4.1. Chapter Overview

This chapter presents the analysis and results from the experiments. The chapter starts with a brief investigation of the overall dataset followed by a discussion of the chosen settings used for the MATLAB Neural Network tool. The rest of the chapter consists of the results and interpretation for each of the datasets used during experimentation.

Later in this chapter much effort goes into gleaning as much information as possible, through neural network analysis, from the datasets discussed in Chapter III. In order to better understand those datasets and the results of that analysis, a brief investigation of the overall dataset is presented. Because each of the experimental datasets was created from this overall dataset, the results of the investigation will, in general, apply to all the datasets. The investigation looked at the class breakdown, correlation information of the features, and the dimensionality of the dataset.

4.2. Overall Dataset

The overall analyzable dataset consists of 204,371 observations. Approximately 8% of those observations are labeled as a threat (Low, Medium, or High). A visual representation of the frequency of the threats is shown in Figure 4.1. Although the large gap between the number of threat observations and non-threat observations is likely a realistic expectation for what a normal government network encounters, this overloading of one class of data does not work well with neural network analysis. It causes the neural network output to be heavily skewed in favor of the overloaded class. As previously discussed in Chapter III, three smaller, balanced, datasets were created to address the

unequal class representation. Information about the three experimental datasets can be found in Table 3.3 in Chapter III.

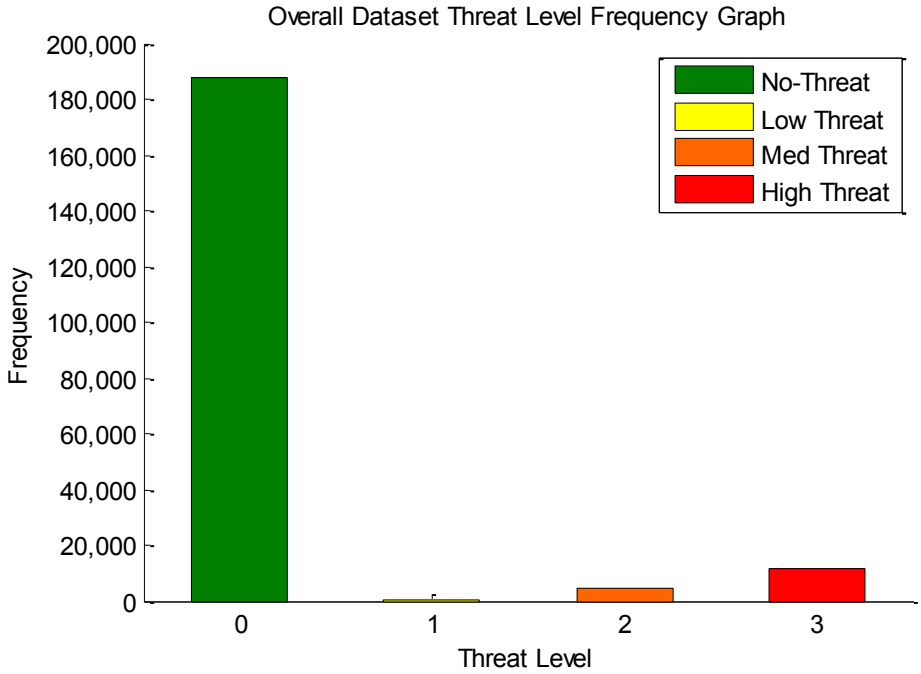

Figure 4.1: Overall Dataset Threat Frequency

Looking at data correlation is an important starting point when investigating datasets. Looking at the correlation can reveal dependence, or relation, among the features and provide some insight into what features do or do not provide additional information (e.g., redundancy). Correlation is used, as opposed to covariance, because with correlation the data is normalized and unit-less, important because the scale varies greatly between the features. Features that are highly correlated with the class feature but not with each other are likely to be the salient features. Due to the high number of features, a color map (see Figure 4.2) is used to visualize the correlation matrix because it is easier to read and interpret than a number matrix would be. As can be seen in the color map, most of the data is uncorrelated (green) with occasional pockets of moderate

(yellow or light blue) to high (red or dark blue) correlation between some features. None of the features seem to be highly correlated with the class (first row/column – somewhat difficult to see); however, there does seem to be some mild to moderate correlation with features numbered in the teens, 80s, 90s, and 170s.

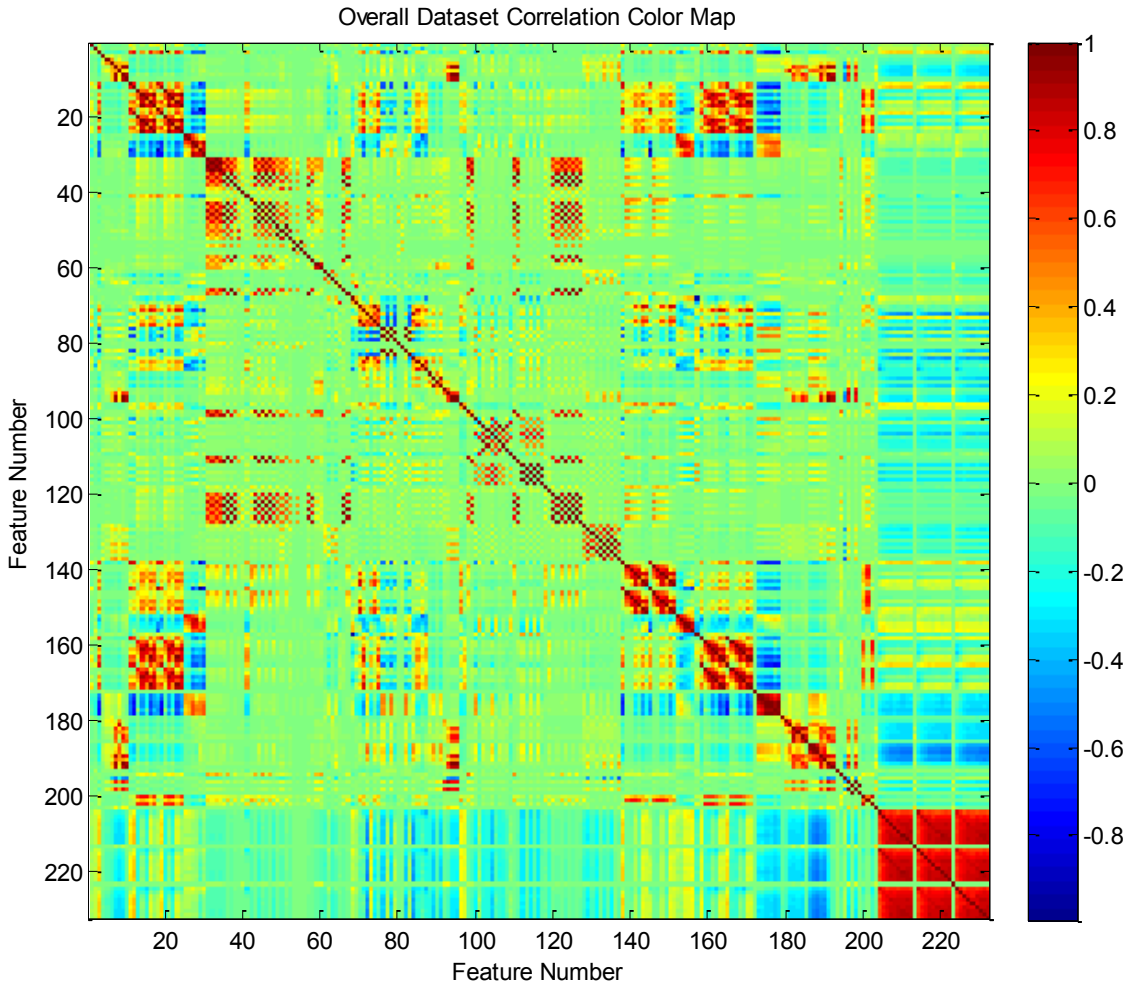

Figure 4.2: Overall Dataset Correlation Color Map

A dimensionality assessment is another way of investigating datasets. Dimensionality assessments provide insight into the number of features (not necessarily which features) that contain information (i.e., explain the variance) about the dataset. A simple dimensionality assessment is Kaiser's Criterion. Figure 4.3 shows a plot of the

Kaiser's Criterion dimensionality test. Using the data correlation, the eigenvalues are computed for each of the features and plotted against the number of features. The number of features with eigenvalues of one or higher should be kept. From the plot we see about 40 features have eigenvalues of 1 or higher. These 40 features explain about 85% of the variance in the data. This indicates the dimensionality of the data is 40 features; however, we have yet to determine which of these features, if any, are salient.

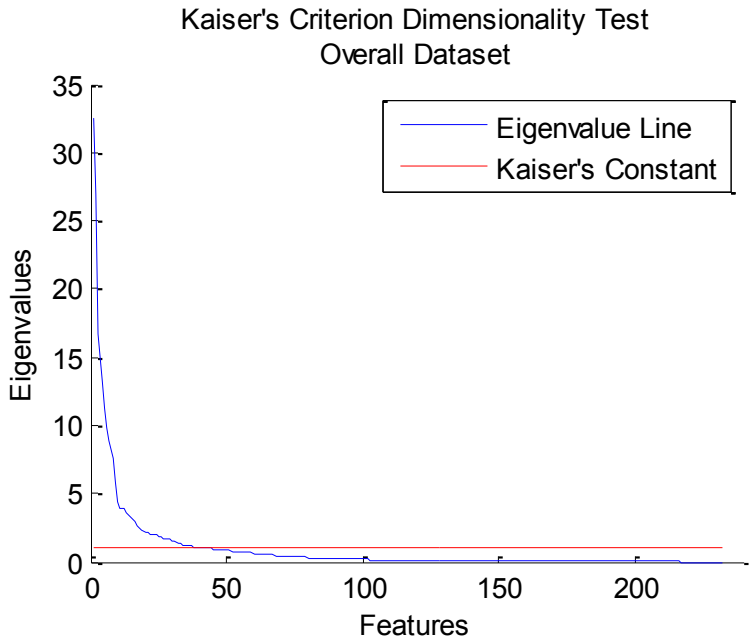

Figure 4.3: Kaiser Dimensionality Plot - Overall Dataset

4.3. MATLAB Neural Network Tool Settings

To maintain consistency throughout experimentation, the same function parameter settings were used for each run of the MATLAB Neural Network pattern recognition tool. Data was divided up randomly using the 'dividerand' setting, selecting 70% of the data for training, 15% for testing, and 15% for training error validation. The transformation function used was the 'logsig' or Log-Sigmoid function. This function was described previously in Chapter III. The Scaled Conjugate Gradient backpropagation

function or ‘trainscg’ was selected as the neural network training algorithm. MATLAB includes several different training algorithms; through initial trial and error, ‘trainscg’ seemed faster than other algorithms with similar classification accuracy outcomes. Network performance was calculated using the cross-entropy algorithm. Typically, mean squared error is the default performance metric, but the natural log function in the cross-entropy algorithm factors in the accuracy prediction values and becomes a more fine-grained method of calculating error thus providing better performing networks [76].

Table 4.1 lists out the parameter settings used to determine when the training tool should end a training iteration. The automation settings described in Chapter III ran the training tool through 30 network-generating attempts for each number of features with network structures containing 10, 20, 30, 40, and 50 hidden nodes. For deciding which class an observation belongs to, the default threshold value of .5 was used for each dataset. After the optimal operating characteristics were determined, the optimal threshold was used for the validations datasets.

Table 4.1: MATLAB Training Tool Settings

Training Tool Settings	Setting
Epochs (max)	500
Time (max)	300 (seconds)
Network Performance Goal	.005
Validation Checks (max)	10

4.4. Dataset Analysis

This section walks through the results from the analysis of the three experimental datasets. Each of the three subsections focuses on a particular dataset, first discussing the chosen hidden layer structure and the overall accuracy leading to the decision on the number of features to keep. Next, the results of the neural network’s performance metrics

and optimal operating characteristics are presented. The data used for these metrics consists of the combination of the training and testing sets calculated using only the retained feature set. This is followed by the performance metrics computed with the neural network using the withheld validation data. Each subsection concludes with a discussion of the salient features for each dataset.

4.4.1. Threat vs. No-Threat Dataset

The first dataset is referred to as the *Threat vs. No-Threat* dataset. This dataset is the most general and, because of that, has the most observations of the three datasets. The dataset is broken down into two classes; an observation is either a threat or not a threat. The analysis for this dataset focuses solely on determining whether or not a threat is present and provides no information on the risk level of the threat discussed. It is likely a method such as this would be used in conjunction with another threat detection method capable of discerning the level of the threat.

4.4.1.1. Hidden Layer Structure – Threat vs. No-Threat Dataset

The number of nodes in the hidden layer makes up the hidden layer structure. The importance of the hidden layer structure is discussed in Chapter III. Each dataset was run through the MATLAB Neural Network tool using 10, 20, 30, 40, and 50 hidden nodes to determine which structure provided the most benefit when factored against the time required to process the data. Plots of the overall accuracy against the number of features (commonly referred to as a "knee plot") and the overall accuracy values were considered for each structure to determine an appropriate performance "drop" point. Table 4.2 presents the results of the testing done with regard to the hidden layer structure. For the *Threat vs. No-Threat* dataset, dropping below .9355 for the last time was chosen as the

comparison drop point. As can be seen in Table 4.2 the selection method results in a different number of features remaining for each hidden layer structure. The idea is to optimize efficiency – minimize time and the number of features while maximizing accuracy. A plot of the number of nodes versus the accuracy at the drop point and the processing time provides an easy way to visualize the tradeoff and can be seen in Figure 4.4. It is obvious that as the number of hidden nodes increases, the accuracy stays relatively the same while the processing time increases. The highlighted row in Table 4.2 shows the chosen structure of 10 nodes with 13 features remaining.

Table 4.2: Hidden Layer Structure – Performance Comparison - Threat vs. No-Threat Dataset

# of Nodes	Time (s)	Time Diff (s)	Accuracy at Drop	Features Remaining
10	81250	0	.9358	13
20	100856	19606	.9357	11
30	136415	55165	.9360	9
40	155990	74740	.9374	6
50	199852	118602	.9377	10

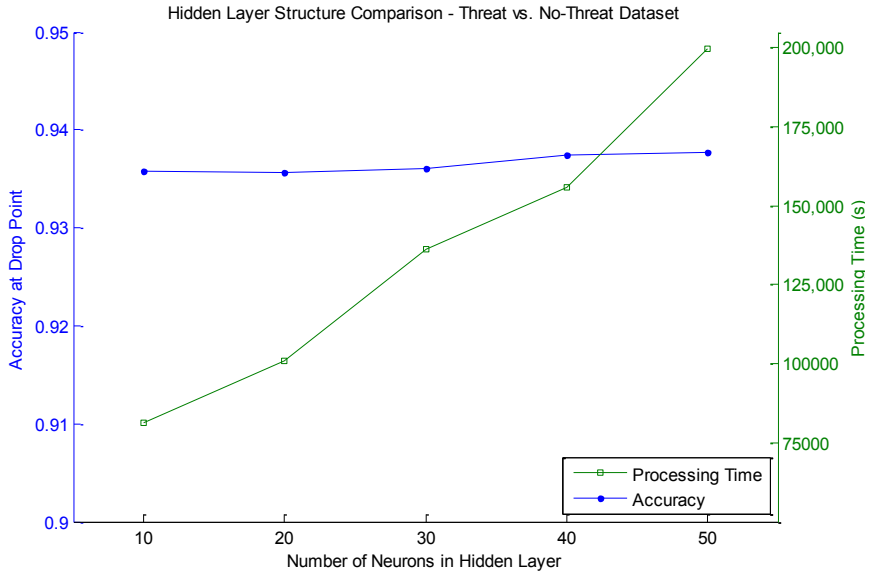

Figure 4.4: Hidden Layer Structure Comparison – Threat vs. No-Threat Dataset

4.4.1.2. Overall Accuracy – Threat vs. No-Threat Dataset

Once the hidden layer structure has been selected, we take a closer look at the performance of the neural network built around the number of remaining features. The objective is to maintain the desired classification accuracy while minimizing the number of features. A knee-plot of the overall accuracy against the number of features removed is shown in Figure 4.5. The accuracy is very consistent at around 94% while most of the features are removed. At feature number 217 (the noted feature), the neural network is performing at a 93.58% classification accuracy rate. After feature number 217 is removed the accuracy starts to take a steep decline and it never comes back up. This is the "knee" point of the plot and determines how many features are required to keep while maintaining the desired classification accuracy. Based on the location of the knee, the decision was made to keep the last 13 features remaining and evaluate the network performance.

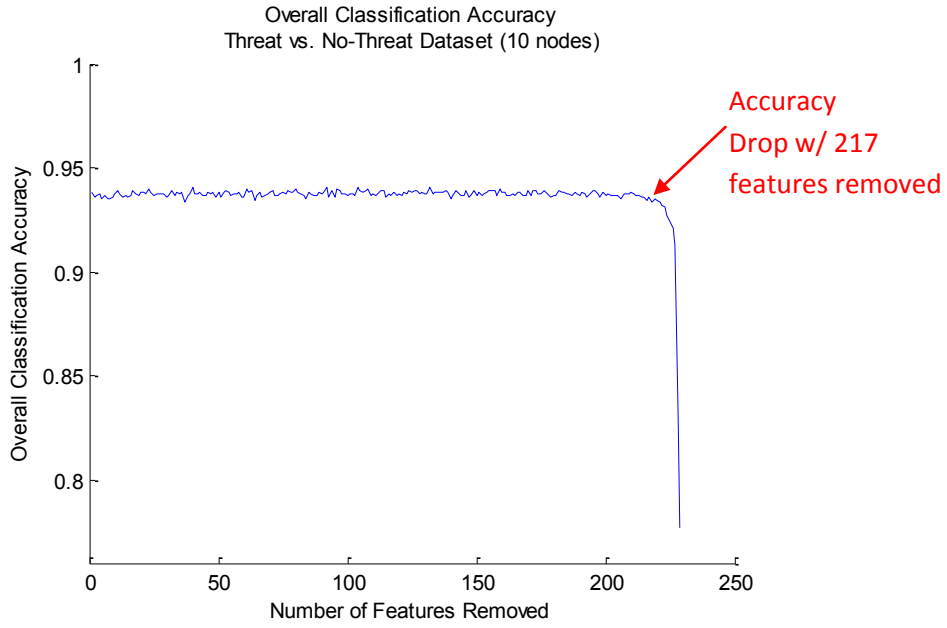

Figure 4.5: Overall Classification Accuracy – Threat vs. No-Threat Dataset (10 nodes)

4.4.1.3. Performance Metrics – Threat vs. No-Threat Dataset

Several performance metrics were calculated to evaluate the neural network created with 13 features remaining. These metrics were calculated from the results shown in the confusion matrix (Table 4.3) and are displayed in Table 4.4. We can see from the results that the generated network performs at over 93% accuracy on the data it was trained and tested on. There is a consistently high classification performance both between classes and within classes (i.e., overall and class-based accuracies).

FPR equates to false alarm rate. The FPRs displayed indicate an expectation of a 2.42% false alarm rate for observations classified as benign and a 9.79% false alarm rate for those classified as a threat. This means we should expect less than 3 out of 100 observations classified as benign to actually be a threat and approximately 10 out of 100 observations classified as a threat to actually be benign.

Table 4.3: Confusion Matrix -
Threat vs. No-Threat Dataset (13 Features)

		Predicted	
		No-Threat	**Threat**
Actual	**No-Threat**	13125	1559
	Threat	326	14359

Table 4.4: Performance Metrics - Threat vs. No-Threat Dataset (13 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
No-Threat	.9358	.9758	.8938	.9021	.0242	.0979	.9339	.9330
Threat	.9358	.9021	.9778	.9758	.0979	.0242	.9392	.9387
Overall Accuracy	.9358							

4.4.1.4. Optimal Operating Characteristics – Threat vs. No-Threat Dataset

The next step in analyzing the outcome of selected neural network was determining the optimal operating characteristics for that network. The optimal operating

characteristics refer to the threshold used when determining what class a particular observation belongs to based on the output score generated by the neural network. The initial decision threshold for all classes in the trained dataset was the default 0.5, considering each dataset equally.

ROC Curves for each class and their associated ensemble threshold plots were generated to check network performance and determine the optimal thresholds and can be seen in Figure 4.6. The ideal location for a ROC curve is the upper left corner of the graph and the lines shown here are very close to that; this indicates the network is performing well. The graph indicates there is a threshold that provides a TPR at or above 0.9 while still keeping an FPR below 0.1 for both target classes. The ensemble threshold plots appear to be robust with threshold values varying between about .15 and .99 for the no-threat class and between about .02 and .85 for the threat class, holding an approximate 90% classification accuracy.

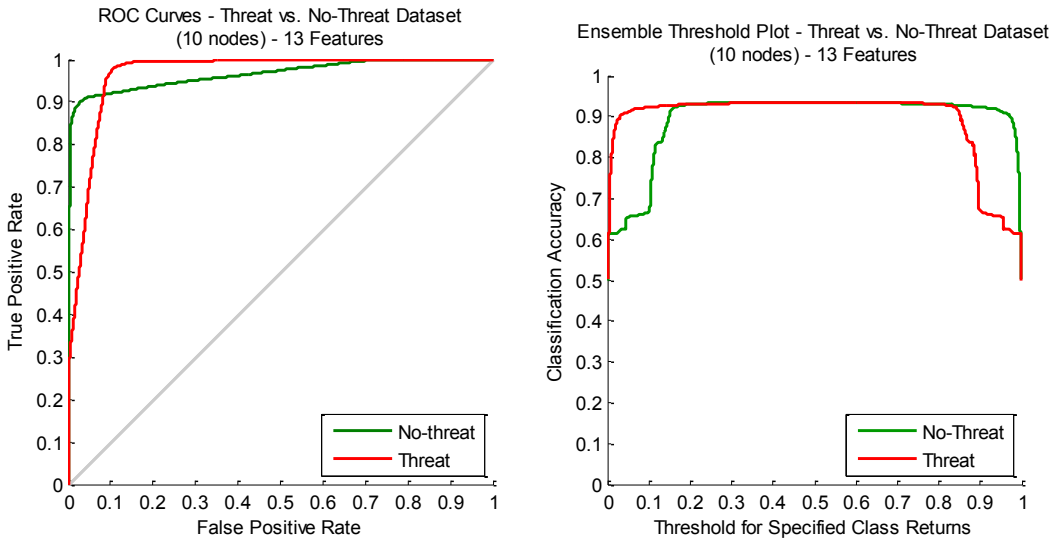

Figure 4.6: ROC Curves and Ensemble Threshold Plots – Threat vs. No-Threat Dataset (10 nodes) - 13 Features

The optimal operating characteristics were explored further looking at the AUC for the ROC curves. The higher the AUC is the higher the predictive capability of the network should be. From Table 4.5 we can see an AUC of .9614 and .9616 for the no-threat and threat classes, respectively. These are high values indicating the network has a high predictive capability. The table also provides the results for the optimal TPR, FPR, and the associated optimal threshold and ensemble accuracy. Notice the similarities between the optimal FPRs and the FPRs shown in Table 4.4. This indicates using the optimal threshold had little effect on the expected false alarm rates of the classes..

Table 4.5: Optimal Operating Characteristics -
Threat vs. No-Threat Dataset (13 Features)

Class	AUC	Optimal TPR	Optimal FPR	Optimal Threshold	Max Ensemble Accuracy
No-Threat	.9614	.8932	.0211	.5449	.9361
Threat	.9616	.9742	.1034	.6099	.9354

4.4.1.5. Validation Results – Threat vs. No-Threat Validation Dataset

Because the generated neural network learns and trains on the input data specifically, it is important to test the network with a separate set of validation data. The optimal thresholds determined in the previous section were used for discriminating between the two classes. Because there are two classes of data, each with its own optimal threshold, the results of testing the validation data consists of two parts, one part focusing on the no-threat class data and the other part focusing on the threat class data.

Table 4.6 provides the confusion matrix results using the .5449 threshold value with a focus on the no-threat class data. The confusion matrix results were then used to calculate the performance metrics for the dataset; those results are shown in Table 4.7. The overall accuracy is above 84% which could be still considered fairly high. The

precision of the threat data at over 96% indicates the neural network predicts the threat data class well, although the FPR indicates that may be due to over-prediction of that class. The effect of the optimal threshold is evident in the associated much lower no-threat data precision result.

We can also see the FPR changes with the validation dataset to 4.29% and 22.15% for the no-threat and threat data classes, respectively. This means about 4 out of 100 observations will be classified as benign when they are actually threats and about 22 out of 100 observations will be classified as a threat when they are actually benign. The false alarm jump for the threat class data from 9.79% with the training data to 22.15% with the validation data is likely due to the neural network overfitting in the training/testing set. In the cyber realm a false alarm rate that high is likely to overwhelm network operators and cause the alert system to be disregarded.

Table 4.6: Confusion Matrix - Threat vs. No-Threat
Validation Dataset - No-Threat Focus (13 Features)

		Predicted	
		No-Threat	**Threat**
Actual	**No-Threat**	1183	449
	Threat	53	1578

Table 4.7: Performance Metrics - Threats vs. No-Threat
Validation Dataset – No-Threat Focus (13 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
No-Threat	.8462	.9571	.7249	.7785	.0429	.2215	.8329	.8250
Threat	.8462	.7785	.9675	.9571	.2215	.0429	.8679	.8628
Overall Accuracy	.8462							

Table 4.8 shows the confusion matrix after using the .6099 threshold, focusing on the data classified as a threat. The network performance metrics were computed from the confusion matrix results and are presented in Table 4.9. The overall accuracy is again

over 84%. Changing the threshold to focus on the threat class data caused little change in the performance of the network, including the false alarm results.

Table 4.8: Confusion Matrix - Threat vs. No-Threat
Validation Dataset - Threat Focus (13 Features)

		Predicted	
		No-Threat	**Threat**
Actual	**No-Threat**	1196	436
	Threat	60	1571

Table 4.9: Performance Metrics - Threats vs. No-Threat
Validation Dataset – Threat Focus (13 Features)

Class	Accuracy	Recall	Precision	Specificity	FPR	FNR	G	F1
No-Threat	.8480	.9522	.7328	.7828	.0478	.2172	.8354	.8283
Threat	.8480	.7828	.9632	.9522	.2172	.0478	.8683	.8637
Overall Accuracy	.8480							

4.4.1.6. Salient Feature Description – Threat vs. No-Threat Dataset

This section provides a description of the 13 salient features chosen for the *Threat vs. No-Threat* dataset. Table 4.10 contains the feature numbers from the original feature set and their associated descriptions. The features are listed by their index number in the original feature set, not according to their weight. The salient features all have to do with the size or number of the packets (also referred to as segments) or the bytes within a packet. Segment size (minimums and maximums) or number of segments is part of 5 of the 13 features. The rest of the features consist of the number of bytes in some portion of the packet, including the control information, which is used to tell the network how and where to deliver the packet and is typically found in the packet headers or trailers.

Table 4.10: Salient Feature Descriptions – Threat vs. No-Threat Dataset [6]

Original Feature Number	Feature description
10	Minimum of bytes in (Ethernet) packet, using the size of the packet on the wire
17	Minimum of total bytes in IP packet, using the size of the payload declared by the IP packet
80	Maximum segment size requested as a TCP option in the SYN packet opening the connection (server to client)
84	Minimum segment size observed during the lifetime of the connection (server to client)
86	Average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data packets reported (server to client)
96	Total number of bytes sent in the initial window (i.e., the number of bytes seen in the initial flight of data before receiving the first ACK packet from the other endpoint acknowledging some data – not the 3-way handshake) (server to client)
97	Total number of segments (packets) sent in the initial window (client to server)
98	Total number of segments (packets) sent in the initial window (server to client)
173	Variance of control bytes packet (client to server)
179	Maximum of bytes in (Ethernet) packet (server to client)
186	Maximum of total bytes in IP packet
187	Variance of total bytes in IP packet (server to client)
193	Maximum of control bytes in packet

4.4.2. Threats Only (Low, Medium, High)

The next dataset is referred to as the *Threats Only* dataset. This dataset consists of only those observations classified as one of the three threat levels, Low, Medium, and High. Because of the limited number of Low threat observations, this dataset is the smallest of the three. Investigation into this dataset is intended to determine how well the neural network can distinguish between the different threat levels so no benign data is included. This type of analysis would work well as a post-investigation to the data characterized as a threat from the previous *Threat vs. No-Threat* dataset.

4.4.2.1. Hidden Layer Structure – Threats Only Dataset

Table 4.11 presents the results of the hidden layer structure performance. For the *Threats Only* dataset, dropping below .95 for the last time was chosen as the comparison drop point. The plot of the number of nodes versus the accuracy at the drop point and the processing time can be seen in Figure 4.7. As the number of hidden nodes increases, the accuracy falls, stays relatively steady, and then drops while the processing time increases. The highlighted row in Table 4.11 shows the chosen structure of 10 nodes with 6 features remaining.

Table 4.11: Hidden Layer Structure – Performance Comparison - Threats Only Dataset

# of Nodes	Time (s)	Time Diff (s)	Accuracy at Drop	Features Remaining
10	2798	0	.9729	6
20	3351	553	.9612	6
30	3699	901	.9612	8
40	3879	1081	.9632	9
50	4183	1385	.9516	8

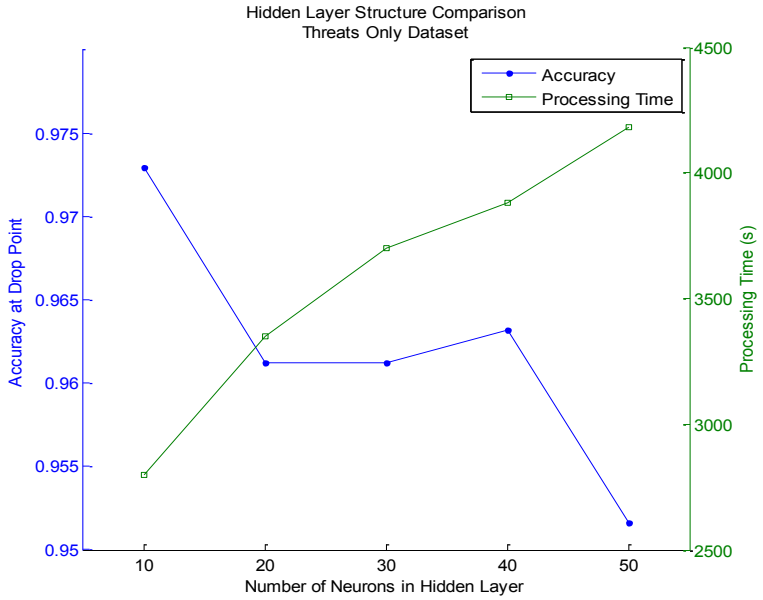

Figure 4.7: Hidden Layer Structure Comparison - Threats Only Dataset

4.4.2.2. Overall Accuracy – Threats Only Dataset

A knee-plot of the overall accuracy against the number of features removed is shown in Figure 4.8. The accuracy fluctuates between about 96% and 99% as most of the features are removed. At feature number 217 (the noted feature), the neural network is performing at a 97.29% classification accuracy rate. After feature number 217 is removed, the accuracy starts to take a steep decline and it never comes back up. Based on the location of the knee, the decision was made to keep the last six features remaining and evaluate the network performance.

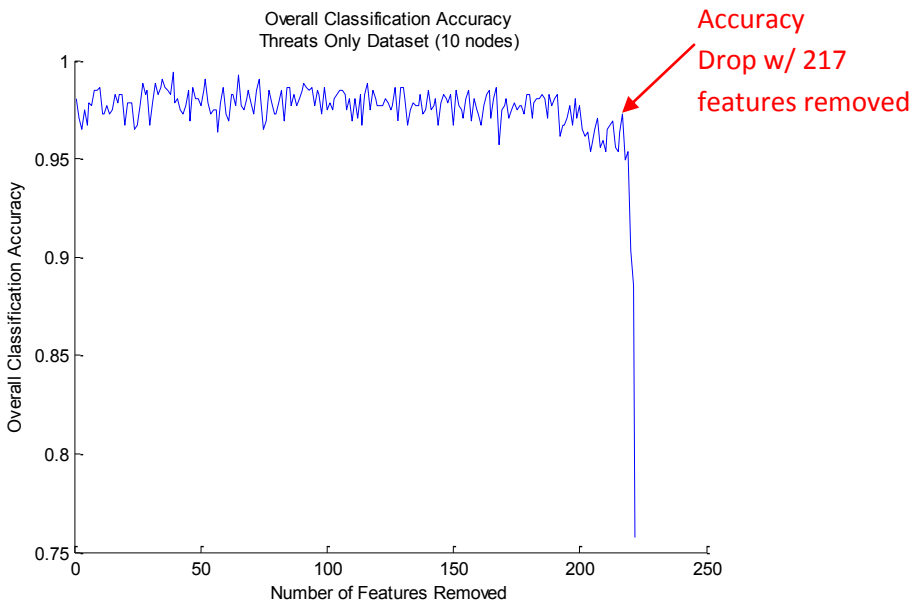

Figure 4.8: Overall Classification Accuracy - Threats Only Dataset (10 nodes)

4.4.2.3. Performance Metrics – Threats Only Dataset

Performance metrics were calculated to evaluate the neural network created with six features remaining. These metrics were calculated from the results shown in the confusion matrix (Table 4.12) and are displayed in Table 4.13. From the results we can see the generated network performs at over 97% accuracy on the data it was trained and

tested on. The high classification performance is consistent both between classes and within classes (i.e., overall and class-based accuracies). Precision for the Low threat class indicates the neural network may not predict that class as well as it does with the other classes. The results for the Medium and High threat classes are similar and very high, suggesting the neural network classifies those classes well.

The FPRs shown indicate an expectation of a 2.58% false alarm rate for observations classified as Low threat, a .88% false alarm rate for those classified as Medium threat, and a .59% false alarm rate for those classified as High threat. This means we should expect approximately 3 out of 100 observations classified as Medium or High threat to actually be Low threat; less than 1 out of 100 observations classified as Low or High threat to actually be Medium threat; and less than 1 out of 100 observations classified as Low or Medium threat to actually be High threat. The false alarm rate for Medium and High threat indicates the network does an excellent job differentiating Medium and High both between each other and against the Low class.

Table 4.12: Confusion Matrix - Threats Only Dataset (6 Features)

		Predicted		
		Low	**Medium**	**High**
Actual	**Low**	163	4	5
	Medium	3	169	0
	High	1	1	170

Table 4.13: Performance Metrics - Threats Only Dataset (6 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
Low	.9748	.9760	.9477	.9742	.0258	.0240	.9618	.9617
Medium	.9845	.9713	.9826	.9912	.0088	.0287	.9769	.9769
High	.9864	.9714	.9884	.9947	.0059	.0286	.9799	.9798
Overall Accuracy		.9729						

4.4.2.4. Optimal Operating Characteristics – Threats Only Dataset

This section discusses the optimal operating characteristics established for the generated neural network. ROC Curves for each class and their associated ensemble threshold plots were generated to check network performance and determine the optimal thresholds. The plots are shown in Figure 4.9. The curve lines shown in the ROC Curve plots are extremely close to the upper left corner indicating a high level of performance from the neural network. From the graph we can see there is a threshold that provides a TPR close to one with an FPR close to zero for all three classes. Similar to the *Threat vs. No-Threat* dataset results, the ensemble threshold plots appear to be quite robust. Threshold values for Low threat class data to achieve at least 90% accuracy range from about 0.1 to .95. Threshold values for Medium and High threat data to get 90% accuracy begin at 0 and extend to 1 and about 0.9 for Medium and High threat data, respectively. These operating characteristic curves reiterate that the generated neural network performs better with Medium and High threat data than it does with the Low threat data.

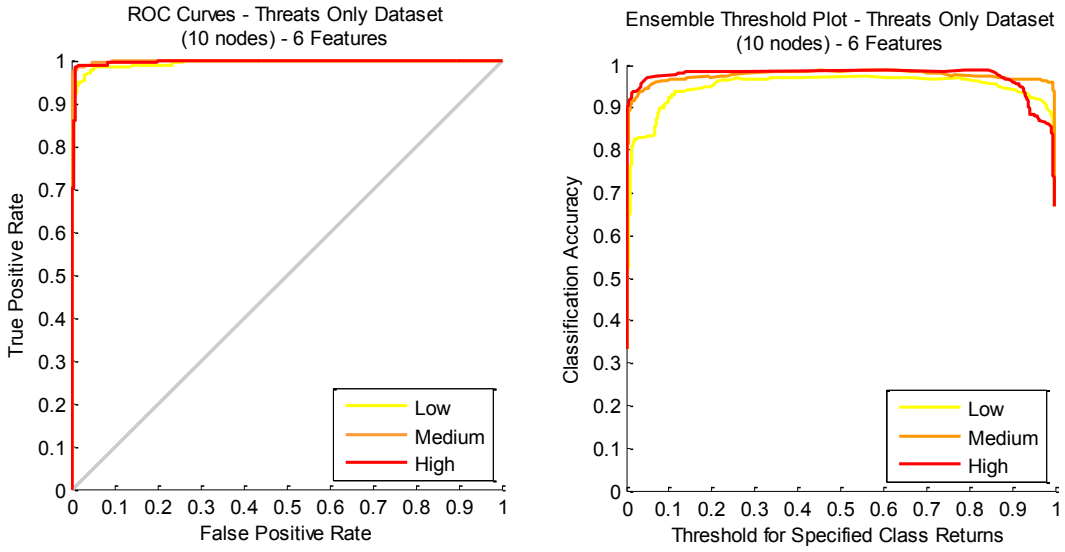

Figure 4.9: ROC Curves and Ensemble Threshold Plots - Threats Only Dataset (10 nodes) - 6 Features

The AUC for the generated ROC curves was also explored. In Table 4.14 we can see an AUC of over .99 for all three classes. These are extremely high values indicating the network has an excellent predictive capability. The optimal TPR, FPR, and associated optimal threshold and ensemble accuracy can also be seen Table 4.14. In contrast to the results from the *Threats vs. No-Threats* dataset, there is little difference between the optimal FPRs and the FPRs shown in Table 4.13. This is likely due to the robustness of the optimal threshold as varying the threshold seems to cause much of an effect on the accuracy outcome. The ranking of the false alarm percentage did swap between the Medium and High threat data but they are both still at less than 1%.

Table 4.14: Optimal Operating Characteristics - Threats Only Dataset (6 Features)

Class	AUC	Optimal TPR	Optimal FPR	Optimal Threshold	Max Ensemble Accuracy
Low	.9931	.9419	.0116	.5627	.9729
Medium	.9988	.9709	.0029	.6213	.9884
High	.9959	.9826	.0087	.8146	.9884

4.4.2.5. Validation Results – Threats Only Validation Dataset

The optimal thresholds determined in the previous section were used for testing the validation data. The following results consist of three parts focusing on each of the Low, Medium, and High threat class data.

Table 4.15 shows the confusion matrix results using the .5627 threshold value with a focus on the Low threat class data. These results were then used to calculate the performance metrics for the dataset which are shown in Table 4.16. The overall accuracy is above 87% which could be still considered fairly high but is a 10 percentage point drop from the training dataset overall accuracy. The precision of the Low threat data at approximately 79%, when compared to the 89% and 95% for the Medium and High

threat data, reinforces the earlier notion that the neural network does not predict the Low threat data class as well as the other classes. The results do show, however, when the focus is on identifying the Low threat, the neural network's recall performance is much better for the Low class than either of the other classes.

We can also see there is a jump in the false alarm rate for all classes with the validation data. The FPR went up to 9.76% for the Low threat class data, 5.41% for the Medium threat class data, and 2.78% for the High threat class data. This translates to an expectation of about 10 out of 100 observations classified as Medium or High threat when they are actually Low; about 5 out of 100 observations classified as Low or High when they are actually Medium; and about 3 out of 100 observations classified as Low or Medium threat when they are actually High. The false alarm jump is again likely due to the neural network overfitting on the training/testing set.

Table 4.15: Confusion Matrix – Threats Only
Validation Dataset – Low Threat Focus (6 Features)

		Predicted		
		Low	**Medium**	**High**
Actual	**Low**	15	3	1
	Medium	0	17	2
	High	1	0	18

Table 4.16: Performance Metrics - Threats Only
Validation Dataset – Low Threat Focus (6 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
Low	.9123	.9375	.7895	.9024	.0976	.0625	.8603	.8571
Medium	.9123	.8500	.8947	.9459	.0541	.1500	.8721	.8718
High	.9298	.8571	.9474	.9722	.0278	.1429	.9011	.9000
Overall Accuracy		.8772						

Table 4.17 shows the confusion matrix after using the .6213 threshold, focusing on the data classified as Medium threat. The network performance metrics were

computed from the confusion matrix results and are presented in Table 4.18. The results are identical to those found when focusing on the Low threat class. Looking at the neural network score output, the scores for each observation are generally high in one class' cell suggesting the network was able to distinguish between the classes at a high level. The change in the threshold was not enough to affect the prediction outcome so the results turned out exactly the same.

Table 4.17: Confusion Matrix – Threats Only
Validation Dataset – Medium Threat Focus (6 Features)

		Predicted		
		Low	**Medium**	**High**
Actual	**Low**	15	3	1
	Medium	0	17	2
	High	1	0	18

Table 4.18: Performance Metrics - Threats Only
Validation Dataset – Medium Threat Focus (6 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
Low	.9123	.9375	.7895	.9024	.0976	.0625	.8603	.8571
Medium	.9123	.8500	.8947	.9459	.0541	.1500	.8721	.8718
High	.9298	.8571	.9474	.9722	.0278	.1429	.9011	.9000
Overall Accuracy		.8772						

Table 4.19 shows the confusion matrix after using the .8146 threshold, focusing on the data classified as High threat. The network performance metrics were computed from the confusion matrix results and are presented in Table 4.20. The overall accuracy stayed exactly the same at 87.72%. Focusing on the High threat class evened out recall and precision for the Low and Medium threat class data. This balancing out was the result of the prediction values for the Low and Medium threat turning out the same with 16 true positives and 3 false positives. The false alarm rate reflected the effect as well with the High threat data remaining at 2.78% and the Low and Medium threat data even at 7.69%.

Table 4.19: Confusion Matrix – Threats Only
Validation Dataset – High Threat Focus (6 Features)

		Predicted		
		Low	**Medium**	**High**
Actual	**Low**	16	2	1
	Medium	1	16	2
	High	1	0	18

Table 4.20: Performance Metrics - Threats Only Validation Dataset –
High Threat Focus (6 Features)

Class	Accuracy	Recall	Precision	Specificity	FPR	FNR	G	F1
Low	.9123	.8889	.8421	.9231	.0769	.1111	.8652	.8649
Medium	.9123	.8889	.8421	.9231	.0769	.1111	.8652	.8649
High	.9298	.8571	.9474	.9722	.0278	.1429	.9011	.9000
Overall Accuracy	.8772							

4.4.2.6. Salient Feature Description – Threats Only Dataset

This section provides a description of the six salient features chosen for the *Threats Only* dataset (see Table 4.21). All of the salient features pertain to segment size or number of bytes in a section of the packet, again, mostly the control section. The one exception is the minimum window advertisement seen. The window advertisement is a flow control mechanism sent from the receiver letting the sender know how much of data can be received before the sender has to wait for an acknowledgment.

Table 4.21: Salient Feature Description - Threats Only Dataset [6]

Original Feature Number	Feature description
81	Maximum segment size observed during the life of the connection (client to server)
83	Minimum segment size observed during the life of the connection (client to server)
90	Minimum window advertisement seen (if both sides negotiated window scaling)(server to client)
171	Third quartile of control bytes in packet
173	Variance of control bytes in packet
180	Variance of bytes in Ethernet packet

4.4.3. Complete Set (None, Low, Medium, High)

The final dataset is the *Complete* dataset. This dataset consists of four classes: None, Low threat, Medium threat, and High threat. The limited number of Low threat observations affects the size of this dataset as it did the *Threats Only* dataset, however, with the inclusion of the None data, this data is slight larger. Investigation into this dataset is intended to test how well the neural network can not only detect a threat, but also determine the level of the threat. Unlike the previous datasets, this method should be able to stand on its own without further processing.

4.4.3.1. Hidden Layer Structure – Complete Dataset

Table 4.22 presents the results of the hidden layer structure performance. For the *Complete* dataset, dropping below .8 for the last time was chosen as the comparison drop point. The plot of the number of nodes versus the accuracy at the drop point and the processing time can be seen in Figure 4.10. Unlike the previous datasets, classification performance jumps as the nodes increase to 30 before dropping rapidly, while incurring only a mild increase in processing time, thus making 30 nodes the chosen structure as opposed to the 10 nodes used for the previous datasets. The highlighted row in Table 4.22 shows the chosen structure of 30 nodes with 8 features remaining.

Table 4.22: Hidden Layer Structure - Performance Comparison - Complete Dataset

# of Nodes	Time (s)	Time Diff (s)	Accuracy at Drop	Features Remaining
10	3573	0	.8110	11
20	3634	61	.8110	10
30	4063	490	.8256	8
40	4149	576	.8125	7
50	4561	988	.8009	7

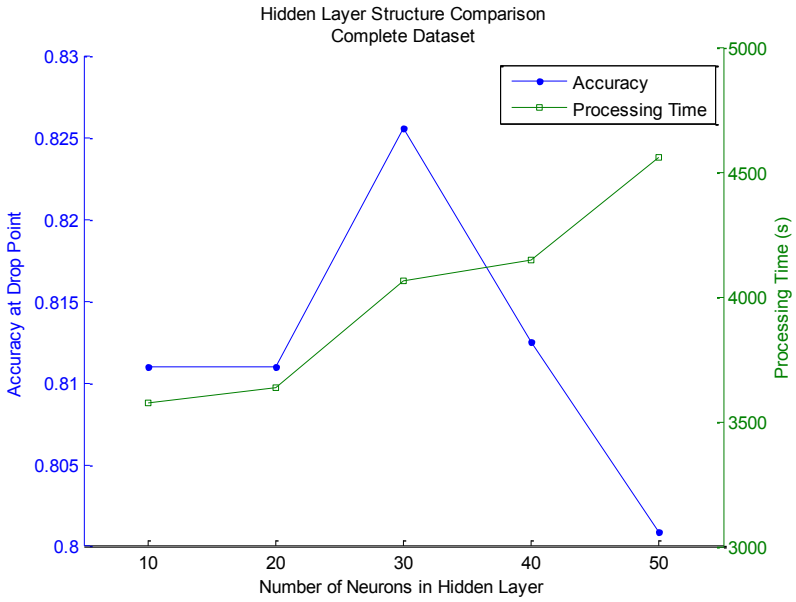

Figure 4.10: Hidden Layer Structure Comparison - Complete Dataset

4.4.3.2. Overall Accuracy – Complete Dataset

A knee-plot of the overall accuracy against the number of features removed is shown in Figure 4.11. The accuracy bounces between about 83% and 87% as most of the features are removed. At this point we see indicators that the neural network does not distinguish threat and threat levels as well as it does when they are separated, as the overall accuracy of the training data is, on average, at least a full 10 percentage points lower than it was for each of the other datasets. At feature number 217 (the noted feature), the neural network is performing at an 82.56% classification accuracy rate. After feature number 217 is removed, the accuracy starts to decline and never comes back up. Based on the location of the knee, the decision was made to keep the last eight features remaining and evaluate the network performance.

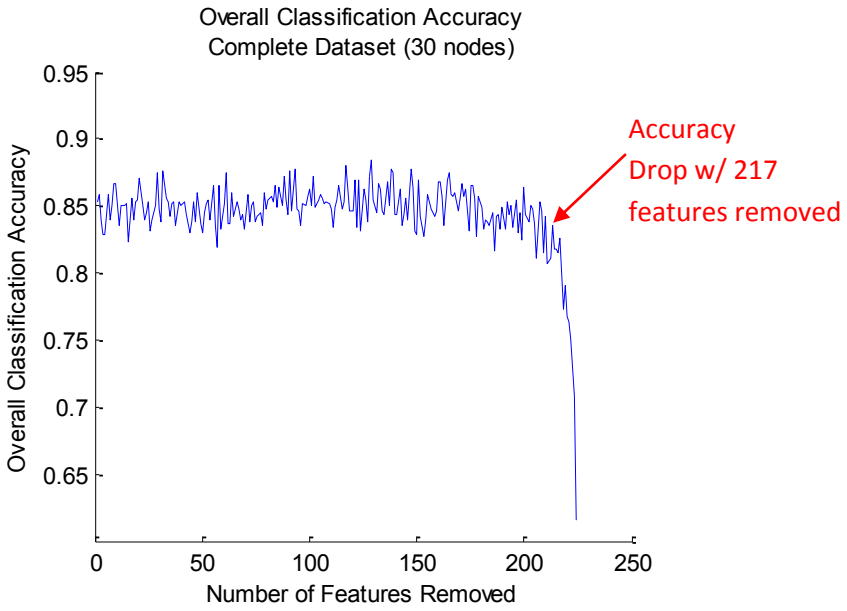

Figure 4.11: Overall Classification Accuracy - Complete Dataset (30 nodes)

4.4.3.3. Performance Metrics – Complete Dataset

Performance metrics were calculated to evaluate the neural network created with eight features remaining. These metrics were calculated from the results shown in the confusion matrix (see Table 4.23) and are displayed in Table 4.24. From the results we can see the generated network performs at almost 83% accuracy on the data it was trained and tested on. The classification performance is inconsistent both between classes and within classes (i.e., overall and class-based accuracies). The None and Low threat classification accuracies are similar to each other but different from the Medium and High threat classification accuracies. The low precision values for the None and Low threat classes, in contrast to the high values for the Medium and High threat classes, indicate the neural network may not predict those classes as well as it does with the other classes. Classification of the Medium threat class seems to generally be the highest across the range of statistics presented.

The FPRs shown indicate an expectation of an 11.28% false alarm rate for observations classified as None, 7.72% false alarm rate for those classified as Low threat, 2.68% false alarm rate for those classified as Medium threat, and 1.04% false alarm rate for those classified as High threat. This means we should expect about 11 out of 100 observations classified as some level of threat to actually be benign; 8 out of 100 observations classified as None, Medium, or High threat to actually be Low threat; 3 out of 100 observations classified as None, Low, or High threat to actually be Medium threat; and 1 out of 100 observations classified as None, Low, or Medium threat to actually be High threat. The network appears to distinguishing the High threat well.

Table 4.23: Confusion Matrix - Complete Dataset (8 Features)

		Predicted			
		None	**Low**	**Medium**	**High**
Actual	**None**	111	29	4	28
	Low	31	132	3	6
	Medium	3	7	158	4
	High	2	2	1	167

Table 4.24: Performance Metrics - Complete Dataset (8 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
None	.8590	.7551	.6453	.8872	.1128	.2449	.6981	.6959
Low	.8866	.7765	.7674	.9228	.0772	.2235	.7719	.7719
Medium	.9680	.9518	.9186	.9732	.0268	.0482	.9351	.9349
High	.9375	.8146	.9709	.9896	.0104	.1854	.8894	.8859
Overall Accuracy		.8256						

4.4.3.4. Optimal Operating Characteristics – Complete Dataset

ROC Curves for each class and their associated ensemble threshold plots were generated to check network performance and determine the optimal thresholds. The plots are shown in Figure 4.12. The curve lines shown in the ROC Curve plots for the Medium and High threat classes are close to the upper left corner indicating a high level of

performance from the neural network. From the graph we can see there is a threshold that provides a TPR greater than .9 with an FPR less than .1 for both classes. The curves for the None and Low threat classes suggest the neural network does not perform as well for classifying those two classes as the FPR needed for a TPR of .9 is close to .3, meaning 30% of the classification would result in false alarms. Similar to the previous dataset results, the ensemble threshold plots appear to be fairly robust, although the achieved accuracy is not as high for the *Complete* dataset. There is an obvious gap between the accuracies of the None and Low threat classes and the Medium and High classes. These operating characteristic curves demonstrate that the generated neural network performs better with Medium and High threat data than it does with the None and Low threat data.

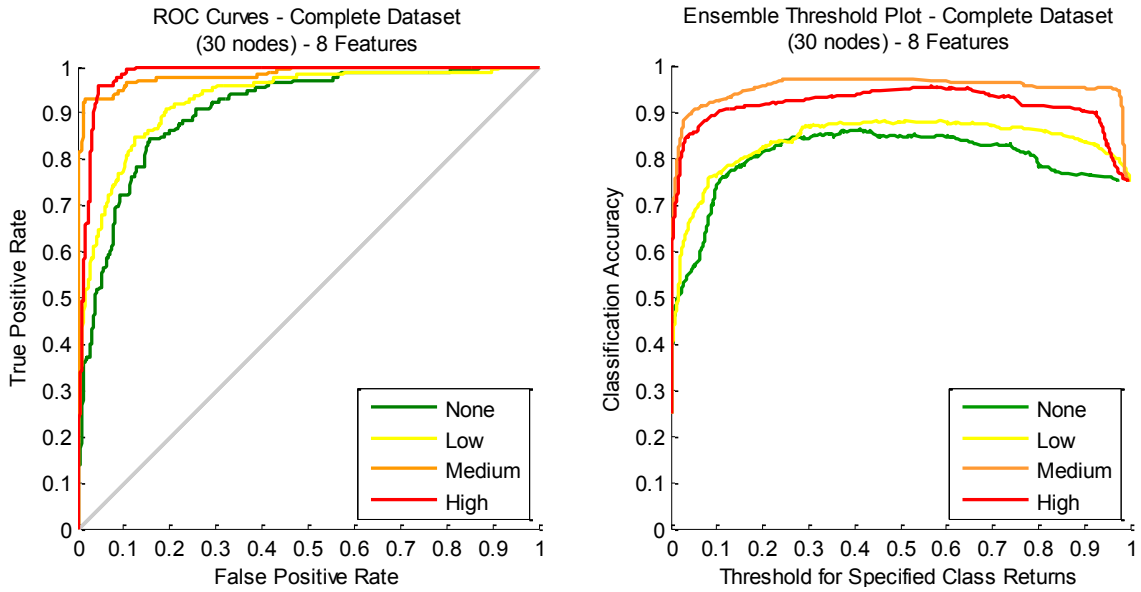

Figure 4.12: ROC Curves and Ensemble Threshold Plots - Complete Dataset (30 nodes) - 8 Features

The AUC for the generated ROC curves was also explored. In Table 4.25 we can see an AUC of over .90 for all four classes. The Medium and High threat class data results show extremely high values of .9838 and .9828, respectively, indicating the neural

network has an excellent predictive capability for those particular classes. The lower AUC values for the None and Low threat classes show the neural network has a good but not great predictive capability. The optimal TPR, FPR, and associated optimal threshold and ensemble accuracy can also be seen in the table. There is a noticeable difference between the optimal FPRs and the FPRs shown in Table 4.24. This difference indicates that, similarly to the *Threat vs. No-Threat* dataset, the threshold has an effect on the false alarm rate.

Table 4.25: Optimal Operating Characteristics - Complete Dataset (8 Features)

Class	AUC	Optimal TPR	Optimal FPR	Optimal Threshold	Max Ensemble Accuracy
None	.9055	.6919	.0814	.4106	.8619
Low	.9287	.6802	.0523	.5854	.8808
Medium	.9838	.9244	.0136	.4022	.9709
High	.9828	.9593	.0465	.5672	.9549

4.4.3.5. Validation Results – Complete Validation Dataset

As with the previous datasets, the optimal thresholds determined in the previous section were used for testing the validation data. The following results consist of four parts focusing on each of the None, Low, Medium, and High threat class data.

Table 4.26 shows the confusion matrix results using the .4106 threshold value with a focus on the None class data. These results were then used to calculate the performance metrics for the dataset which are shown in Table 4.27. The overall accuracy dropped to 65.79%, a reduction by almost 15 percentage points from the training data. Recall is a low 43% and 53% for the None and Low threat class data compared to 100% and 75% for the Medium and High threat data. Precision presents similar results. This

reinforces the earlier notion that the neural network does not predict the None and Low threat data class as well as it does for the other two classes.

The false alarm rate increases for all classes with the validation data; the highest increase is with Low class data which rose over 10 percentage points. Coupled with the FPR of the None class (at almost 17%), the Low class FPR increase likely has to do with the neural network having a difficult time distinguishing between the None and Low threat classes. The results translate to an expectation of about 17 out of 100 observations classified as some threat level when they are actually None; 18 out of 100 observations classified as None, Medium, or High threat when they are actually Low threat; 8 out of 100 observations classified as None, Low, or High threat when they are actually Medium threat; and 2 out of 100 observations classified as None, Low, or Medium threat when they are actually High threat.

Table 4.26: Confusion Matrix – Complete Validation Dataset – No-Threat Focus (8 Features)

		Predicted			
		None	**Low**	**Medium**	**High**
Actual	**None**	10	4	0	5
	Low	10	8	0	1
	Medium	2	3	14	0
	High	1	0	0	18

Table 4.27: Performance Metrics - Complete Validation Dataset – No-Threat Focus (8 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
None	.7105	.4348	.5263	.8302	.1698	.5652	.4784	.4762
Low	.7632	.5333	.4211	.8197	.1803	.4667	.4739	.4706
Medium	.9342	1.000	.7368	.9194	.0806	0	.8584	.8485
High	.9079	.7500	.9474	.9808	.0192	.2500	.8429	.8372
Overall Accuracy	.6579							

Table 4.28 shows the confusion matrix after using the .5854 threshold, focusing on the data classified as Low threat. Table 4.29 contains the performance metrics computed from the confusion matrix results. The overall accuracy dropped slightly from the None class focused value to 63.16%. Focusing on the Low threat reduced the neural network's performance on all four of the classes. This reduction in performance includes the false alarm rate with FPR staying the same or increasing for all classes.

Table 4.28: Confusion Matrix – Complete Validation Dataset – Low Threat Focus (8 Features)

		Predicted			
		None	**Low**	**Medium**	**High**
Actual	**None**	10	4	0	5
	Low	11	7	0	1
	Medium	2	3	14	0
	High	2	0	0	17

Table 4.29: Performance Metrics - Complete Validation Dataset – Low Threat Focus (8 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
None	.6842	.4000	.5263	.8235	.1765	.6000	.4588	.4545
Low	.7500	.5000	.3684	.8065	.1935	.5000	.4292	.4242
Medium	.9342	1.000	.7368	.9194	.0806	0	.8584	.8485
High	.8947	.7391	.8947	.9623	.0377	.2609	.8132	.8095
Overall Accuracy		.6316						

Table 4.30 shows the confusion matrix after using the .4022 threshold, focusing on the data classified as Medium threat; the associated performance metrics are presented in Table 4.31. The results are identical to those found when focusing on the None class; this is not surprising with the threshold values being so similar (.4106 vs. .4022). The change in the threshold is not large enough to create a change in the classification of the neural network's output scores.

Table 4.30: Confusion Matrix – Complete Validation Dataset - Medium Threat Focus (8 Features)

		Predicted			
		None	**Low**	**Medium**	**High**
Actual	**None**	10	4	0	5
	Low	10	8	0	1
	Medium	2	3	14	0
	High	1	0	0	18

Table 4.31: Performance Metrics - Complete Validation Dataset – Medium Threat Focus (8 Features)

Class	**Accuracy**	**Recall**	**Precision**	**Specificity**	**FPR**	**FNR**	**G**	**F1**
None	.7105	.4348	.5263	.8302	.1698	.5652	.4784	.4762
Low	.7632	.5333	.4211	.8197	.1803	.4667	.4739	.4706
Medium	.9342	1.000	.7368	.9194	.0806	0	.8584	.8485
High	.9079	.7500	.9474	.9808	.0192	.2500	.8429	.8372
Overall Accuracy	.6579							

Table 4.32 shows the confusion matrix after using the .5672 threshold, with a focus on the data classified as High threat. The neural network performance metrics were computed from the confusion matrix results and are presented in Table 4.33. Focusing on the High threat reduced the neural network's performance on all four of the classes. The reduction in performance includes the false alarm rate with FPR staying the same or increasing for all classes.

Table 4.32: Confusion Matrix – Complete Validation Dataset – High Threat Focus (8 Features)

		Predicted			
		None	**Low**	**Medium**	**High**
Actual	**None**	10	4	0	5
	Low	11	7	0	1
	Medium	2	3	14	0
	High	1	0	0	18

Table 4.33: Performance Metrics - Complete Validation Dataset – High Threat Focus (8 Features)

Class	Accuracy	Recall	Precision	Specificity	FPR	FNR	G	F1
None	.6974	.4167	.5263	.8269	.1731	.5833	.4683	.4651
Low	.7500	.5000	.3684	.8065	.1935	.5000	.4292	.4242
Medium	.9342	1.000	.7386	.9194	.0806	0	.8584	.8485
High	.9079	.7500	.9474	.9808	.0192	.2500	.8429	.8372
Overall Accuracy	.6447							

Varying the threshold for this dataset seems to have little effect; it either reduced or maintained the performance results. The little variance in the optimal threshold values indicates the neural network output scores for the *Complete* dataset must also have little variance making it more difficult to distinguish between them.

4.4.3.6. Salient Feature Description – Complete Dataset

This section provides a description of the eight salient features chosen for the *Complete* dataset. Table 4.34 contains the feature numbers from the original feature set and their associated descriptions.

Table 4.34: Salient Features - Complete Dataset [6]

Original Feature Number	Feature description
17	Minimum of total bytes in IP Packet, using the size of the payload declared by the IP Packet
26	Median of control bytes in packet
86	Average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data packets reported (server to client)
90	Minimum window advertisement seen (if both sides negotiated window scaling)(server to client)
158	Maximum of bytes in (Ethernet) packet
173	Variance of control bytes in packet
178	Third quartile of bytes in (Ethernet) packet
187	Variance of total bytes in IP packet

Similarly to the previously analyzed datasets, the salient features reference the number of bytes in certain sections of the packet (Ethernet and IP), segment size feature, and the minimum window advertisement seen.

4.5. Summary

This chapter presented the results and analysis for the experimentation done with the three datasets developed in Chapter III. Neural networks were generated for each dataset. The best network for each dataset was chosen based on a combination of a heuristically chosen hidden layer structure and the overall accuracy percentage drop as features were removed. The *Threat vs. No-Threat* dataset exploration resulted in a hidden layer structure containing 10 nodes, with 13 features retained. The *Threats Only* dataset's chosen structure contained 10 nodes as well, with 6 features retained. The *Complete* dataset, encompassing both the benign and threat-level distinguish data, resulted in a 30 node hidden layer and 8 features retained.

Performance metrics were calculated for each dataset's chosen neural network. The *Threats Only* dataset had the best overall classification accuracy with 97.29%, followed by the *Threat vs. No-Threat* dataset with 93.58%, and the *Complete* dataset with 82.56%. An examination of the optimal operating characteristics using ROC curves and ensemble threshold plots resulted in a similar ranking between the three datasets.

The results from the optimal operating characteristics were used with the validation datasets, the 10% withheld from the original dataset, to test the general predictive capabilities of the selected neural networks for each dataset. The edge for overall classification accuracy went to the *Threats Only* dataset with the *Threat vs. No-Threat* dataset following behind. The results from analysis of the *Complete* dataset show

a distinct weakness in the generated neural network's predictive capability for distinguishing across the four classes of data. The high classification performance shown for the Medium and High threat classes is countered by the mediocre performance shown for the None and Low threat classes.

False alarm rates for each class in each dataset were also explored. The *Threat vs. No-Threat* and *Complete* datasets presented the highest false alarm rates, especially for the validation data. This was not surprising considering the neural networks for those datasets performed the worst. The high false positive rates could possibly be mitigated through use of a secondary classification or ensemble method.

Finally, the salient features of each dataset were discussed. One feature, number 173 (variance of control bytes in packet*)* appeared in all three datasets. Numbers 17 (Minimum of total bytes in IP Packet, using the size of the payload declared by the IP Packet), 86 (Average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data packets reported (server to client)), 90 (Minimum window advertisement seen (if both sides negotiated window scaling) (server to client)), and 187 (Variance of total bytes in IP packet (server to client)) were each seen in 2 of the datasets. The rest of the features shared segment size or a count of the number of bytes in sections of the packets in common.

V. Conclusion

5.1. Chapter Overview

This chapter provides the key elements derived from this research. Next it discusses how this research may contribute to both the operations research and cyber defense fields. The chapter concludes with some thoughts on potential future research.

5.2. Conclusions of Research

This research determined that 21 of the original 248 features were salient to classifying computer network threats. Common components of these salient features included segment size (maximum and minimum), number of segments or bytes sent in the initial window, the minimum window advertisement seen, and a count of the number of bytes in Ethernet, IP, or control packet sections (maximum, minimum, quartiles, total, and variance). Table 5.1 lists the 21 salient features. This list combines those features deemed salient from all three datasets.

Considering the features by their associated category provides insight into where the salient information resides. Almost half of the features (10 of 21) are part of the Transport (typically TCP) section of the packet. This is slightly deceptive, however, if the correlation between the features is examined. Features 81 and 83 share a 97.8% correlation. This is not surprising as the features are either minimums or maximums of the same information (segment size from client to server). It is likely only one of each of those features is necessary to acquire the available information. A similar situation occurs with features 80 and 90 with a correlation of 75.9%.

Table 5.1: Salient Feature Categorization [6]

Original Feature Number	Feature description	Category
26	Median of control bytes in packet	Transport
80	Maximum segment size requested as a TCP option in the SYN packet opening the connection (server to client)	
81	Maximum segment size observed during the life of the connection (client to server)	
83	Minimum segment size observed during the life of the connection (client to server)	
84	Minimum segment size observed during the lifetime of the connection (server to client)	
86	Average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data packets reported (server to client)	
90	Minimum window advertisement seen (if both sides negotiated window scaling)(server to client)	
171	Third quartile of control bytes in packet (client to server)	
173	Variance of control bytes in packet(client to server)	
193	Maximum of control bytes in packet (server to client)	
96	Total number of bytes sent in the initial window (i.e., the number of bytes seen in the initial flight of data before receiving the first ACK packet from the other endpoint acknowledging some data – not the 3-way handshake) (server to client)	Initial Window
97	Total number of segments (packets) sent in the initial window (client to server)	
98	Total number of segments (packets) sent in the initial window (server to client)	
10	Minimum of bytes in (Ethernet) packet, using the size of the packet on the wire	Ethernet
158	Maximum of bytes in (Ethernet) packet(client to server)	
178	Third quartile of bytes in (Ethernet) packet(server to client)	
179	Maximum of bytes in (Ethernet) packet (server to client)	
180	Variance of bytes in Ethernet packet (server to client)	
17	Minimum of total bytes in IP Packet, using the size of the payload declared by the IP Packet	IP
186	Maximum of total bytes in IP packet (server to client)	
187	Variance of total bytes in IP packet (server to client)	

This correlation is also not surprising as they are both server-based functions to manage traffic flow (feature 80 synchronizes the initial segment size while feature 90 manages how large each segment size thereafter is). Keeping only one of each of the highly correlated features reduces the number of Transport features to 8, which is still close to double the number of features contained in the other categories.

Another high correlation within a category occurs between features 179 and 180, in the Ethernet category, with a correlation of 94.1%. The Ethernet wrapper encompasses the entire packet so it is understandable that the size of the packet would be a threat indicator. Similarly to the highly correlated features in the Transport category, the correlation value between the feature 179 and 180 suggests keeping only 1 of them would still provide the available information.

A between category correlation of 83.3% occurs between feature 10 in the Ethernet category and feature 17 in the IP category. Ethernet wraps around the IP part of the packet so it is not surprising that the number of bytes is correlated between the two.

The other category of packet information shown here is the Initial Window. The initial window consists of the initial data sent (after the three-way handshake establishing the communication link) from one end point to another before the first acknowledgment is received by the sender. The initial window's inclusion as salient points out that threat information is likely to appear in the first round of data passing between end points.

The outcome of this research reveals that the neural networks generated in this research seem to be best suited for distinguishing whether or not a threat exists or, if a threat exists, what risk level the threat is. When the two concepts are combined the network performance suffers. The salient features are contained in four general categories

of packet or flow information: Transport, Initial Window, Ethernet, and IP. Most of the features fall under the Transport category and, especially when size is the metric, end up affecting the Ethernet and IP values because of the packet structure. Segment size or number of bytes seems to have the highest effect on threat classification. Taking feature correlation greater than 80% into consideration, the original list of 248 features can be pruned down to 18 features while still retaining enough information to detect threats with high accuracy.

5.3. Research Contributions

This research makes its contributions in two ways. The first way is providing insight into what components of network traffic should be focused on when trying to detect and classifying potential threats. The magnitude of network traffic information is overwhelming and most of it is likely unimportant. Knowing what areas to focus on allows for faster, more efficient, processing and hopefully, better protection against any potential threats.

The second contribution is the less obvious but still important demonstration of combining the field of OR with the field of cyber operations. The developmental process for this thesis provides testimony to the value of multidisciplinary OR personnel. Familiarity with computer programming and computer communication networks allowed for the data processing and, after applying the OR tools, comprehension and interpretation of the results. Without knowledge and application of concepts in both fields, this research could not have been done. Members of the OR field should be encouraged to gain expertise in other fields to see what OR tools and techniques can be applied and new information discovered.

5.4. Recommendations for Future Research

The research in this thesis is limited to the scope time and available resources allowed. Other aspects of the research may be worthy considering for future research. One future consideration could be cost of misclassification. The costs of miscalculation were considered equally for this research (i.e., there was no difference in the penalty for failing to classify any of the classes). It might be beneficial, considering the potential detrimental effects of a successful Medium or High threat intrusion, to conduct an analysis of the salient features weighting the costs of miscalculating the Medium or High threat observations heavier than that of the benign or Low threat observations.

Another consideration for future research could be looking at online, or real-time, versus offline classification. The research in this thesis was conducted on data that was previously captured. Discovering a threat offline provides little opportunity to prevent intrusion – it is reactive as opposed to proactive. Determining which features work best for a real-time analysis could enable better computer network protection, especially if combined with reduced resources necessary if only the first few packets of a flow are needed.

A final future research consideration could be exploring the different training algorithms available for training and evaluating neural network performance. The algorithms and methods chosen for this research were chosen mainly due to time constraints. It is possible some of the other training algorithms may train better performing networks and produce improved results.

5.5. Summary

This research examined computer network traffic to determine what features of the traffic were salient to detecting and classifying threats. The data captured from the CDX was converted to a dataset with 248 features which was then separated into 3 smaller, specifically designed datasets. These datasets were reduced, through the use of neural networks, to sets ranging from 6 to 13 features. The combined number of features totaled 21, although looking at the correlation between the features revealed the possible presence of redundancy. The generated neural networks performed at a high level when either detecting threats or distinguishing between them, but performance suffered when combining both concepts. Four categories of packet information emerged as being salient to threat detection and classification: Transport, Initial Window, Ethernet, and IP.

Tactics and techniques of network attack change and adapt over time. Because the dataset used in this research spanned seven years, the results show temporal stability in the outcome. It is quite possible, however, this could change though as new protocols emerge (e.g., IPv6) and new technologies offer those with malicious intent new ways of accessing computer networks. Like the potential attackers, the protection must adapt and examination of the salient features should be done periodically to discover any changes.

Appendix A: Acronym List

Acronym	Definition
ACK	Acknowledge
AFIT	Air Force Institute of Technology
ANN	Artificial Neural Network
AUC	Area under the curve
BayesNet	Bayesian Network
BoF	Bag-of-Flows
BoW	Bag-of-Words
C4.5	C4.5 Decision Tree
CBA	Classification-Based Association
CDX	Cyber Defense Exercise
CFS	Correlation-based algorithm
CMAT	Classification-Based on Multiple Association Rules
CON	Consistency-based algorithm
CPAR	Classification-Based on Predictive Association Rules
CSV	Comma Separated Values
DBSCAN	Density-based Spatial Clustering of Applications with Noise
DoD	Department of Defense
EM	Expectation Maximization
FBI	Federal Bureau of Investigation
FCBF	Fast Correlation Based Filter
FIN	Final
FN	False Negative

FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
GECO	Graduate Education Cyberspace Operations
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Message Control Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
kNN	k-Nearest Neighbor
LERAD	Learning Rules for Anomaly Detection
LISSARD	Laboratory for Information System Security/Assurance Research and Development
ML	Machine Learning
MLP	Multilayer Perceptron
MRMR	Maximum Redundancy – Maximum Relevance
NB	Naïve Bayes
NBD	Naïve Bayes Discretisation
NBK	Naïve Bayes Kernel Density Estimation
NBTree	Naïve Bayes Tree
NPS	Naval Postgraduate School
NSA	National Security Agency
OR	Operations Research

OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
PCA	Principal Component Analysis
PDF	Probability Density Functions
POP3	Post Office Protocol
QoS	Quality of Service
RAM	Random Access Memory
ROC	Receiver Operating Characteristic
SFS	Sequential Forward Selection
SMTP	Simple Mail Transfer Protocol
SNR	Signal-to-Noise Ratio
SVM	Support Vector Machines
SYN	Synchronization
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TPR	True Positive Rate
UDP	User Datagram Protocol
US	United States
VBA	Visual Basic for Applications
VPN	Virtual Private Network

Appendix B: Original Feature List

The original 248 feature set from Moore et al. [6].

Number	Short	Long
1	Server Port	Port Number at server; we can establish server and client ports as we limit ourselves to flows for which we see the initial connection set-up.
2	Client Port	Port Number at client
3	min_IAT	Minimum packet inter-arrival time for all packets of the flow (considering both directions).
4	q1_IAT	First quartile inter-arrival time
5	med_IAT	Median inter-arrival time
6	mean_IAT	Mean inter-arrival time
7	q3_IAT	Third quartile packet inter-arrival time
8	max_IAT	Maximum packet inter-arrival time
9	var_IAT	Variance in packet inter-arrival time
10	min_data_wire	Minimum of bytes in (Ethernet) packet, using the size of the packet *on the wire.*
11	q1_data_wire	First quartile of bytes in (Ethernet) packet
12	med_data_wire	Median of bytes in (Ethernet) packet
13	mean_data_wire	Mean of bytes in (Ethernet) packet
14	q3_data_wire	Third quartile of bytes in (Ethernet) packet
15	max_data_wire	Maximum of bytes in (Ethernet) packet
16	var_data_wire	Variance of bytes in (Ethernet) packet
17	min_data_ip	Minimum of total bytes in IP packet, using the size of payload declared by the IP packet
18	q1_data_ip	First quartile of total bytes in IP packet
19	med_data_ip	Median of total bytes in IP packet
20	mean_data_ip	Mean of total bytes in IP packet
21	q3_data_ip	Third quartile of total bytes in IP packet
22	max_data_ip	Maximum of total bytes in IP packet
23	var_data_ip	Variance of total bytes in IP packet
24	min_data_control	Minimum of control bytes in packet, size of the (IP/TCP) packet header
25	q1_data_control	First quartile of control bytes in packet
26	med_data_control	Median of control bytes in packet
27	mean_data_control	Mean of control bytes in packet
28	q3_data_control	Third quartile of control bytes in packet
29	max_data_control	Maximum of control bytes in packet
30	var_data_control	Variance of control bytes packet
31	total_packets_*a b*	The total number of packets seen (client→server).
32	total_packets_*b a*	” (server→client)

Continued on next page

Number	Short	Long
33	ack_pkts_sent_*a b*	The total number of ack packets seen (TCP segments seen with the ACK bit set) (client→server).
34	ack_pkts_sent_*b a*	” (server→client)
35	pure_acks_sent_*a b*	The total number of ack packets seen that were not piggy-backed with data (just the TCP header and no TCP data payload) and did not have any of the SYN/FIN/RST flags set (client→server)
36	pure_acks_sent_*b a*	” (server→client)
37	sack_pkts_sent_*a b*	The total number of ack packets seen carrying TCP SACK [6] blocks (client→server)
38	sack_pkts_sent_*b a*	” (server→client)
39	dsack_pkts_sent_*a b*	The total number of sack packets seen that carried duplicate SACK (D-SACK) [7] blocks. (client→server)
40	dsack_pkts_sent_*b a*	” (server→client)
41	max_sack_blks/ack_*a b*	The maximum number of sack blocks seen in any sack packet. (client→server)
42	max_sack_blks/ack_*b a*	” (server→client)
43	unique_bytes_sent_*a b*	The number of unique bytes sent, i.e., the total bytes of data sent excluding retransmitted bytes and any bytes sent doing window probing. (client→server)
44	unique_bytes_sent_*b a*	” (server→client)
45	actual_data_pkts_*a b*	The count of all the packets with at least a byte of TCP data payload. (client→server)
46	actual_data_pkts_*b a*	” (server→client)
47	actual_data_bytes_*a b*	The total bytes of data seen. Note that this includes bytes from retransmissions / window probe packets if any. (client→server)
48	actual_data_bytes_*b a*	” (server→client)
49	rexmt_data_pkts_*a b*	The count of all the packets found to be retransmissions. (client→server)
50	rexmt_data_pkts_*b a*	” (server→client)
51	rexmt_data_bytes_*a b*	The total bytes of data found in the retransmitted packets. (client→server)
52	rexmt_data_bytes_*b a*	” (server→client)
53	zwnd_probe_pkts_*a b*	The count of all the window probe packets seen. (Window probe packets are typically sent by a sender when the receiver last advertised a zero receive window, to see if the window has opened up now). (client→server)
54	zwnd_probe_pkts_*b a*	” (server→client)
55	zwnd_probe_bytes_*a b*	The total bytes of data sent in the window probe packets. (client→server)
56	zwnd_probe_bytes_*b a*	” (server→client)
57	outoforder_pkts_*a b*	The count of all the packets that were seen to arrive out of order. (client→server)

Continued on next page

Number	Short	Long
58	outoforder_pkts_*b a*	” (server→client)
59	pushed_data_pkts_*a b*	The count of all the packets seen with the PUSH bit set in the TCP header. (client→server)
60	pushed_data_pkts_*b a*	” (server→client)
61	SYN_pkts_sent_*a b*	The count of all the packets seen with the SYN bits set in the TCP header respectively (client→server)
62	FIN_pkts_sent_*a b*	The count of all the packets seen with the FIN bits set in the TCP header respectively (client→server)
63	SYN_pkts_sent_*b a*	The count of all the packets seen with the SYN bits set in the TCP header respectively (server→client)
64	FIN_pkts_sent_*b a*	The count of all the packets seen with the FIN bits set in the TCP header respectively (server→client)
65	req_1323_ws_*a b*	If the endpoint requested Window Scaling/Time Stamp options as specified in RFC 1323[8] a ‘Y’ is printed on the respective field. If the option was not requested, an ‘N’ is printed. For example, an “N/Y” in this field means that the window-scaling option was not specified, while the Time-stamp option was specified in the SYN segment. (client→server)
66	req_1323_ts_*a b*	. . .
67	req_1323_ws_*b a*	If the endpoint requested Window Scaling/Time Stamp options as specified in RFC 1323[8] a ‘Y’ is printed on the respective field. If the option was not requested, an ‘N’ is printed. For example, an “N/Y” in this field means that the window-scaling option was not specified, while the Time-stamp option was specified in the SYN segment. (client→server)
68	req_1323_ts_*b a*	. . .
69	adv_wind_scale_*a b*	The window scaling factor used. Again, this field is valid only if the connection was captured fully to include the SYN packets. Since the connection would use window scaling if and only if both sides requested window scaling [8], this field is reset to 0 (even if a window scale was requested in the SYN packet for this direction), if the SYN packet in the reverse direction did not carry the window scale option. (client→server)
70	adv_wind_scale_*b a*	” (server→client)
71	req sack_*a b*	If the end-point sent a SACK permitted option in the SYN packet opening the connection, a ‘Y’ is printed; otherwise ‘N’ is printed. (client→server)
72	req sack_*b a*	” (server→client)
73	sacks_sent_*a b*	The total number of ACK packets seen carrying SACK information. (client→server)
74	sacks_sent_*b a*	” (server→client)

Continued on next page

Number	Short	Long
75	urgent_data_pkts_*a b*	The total number of packets with the URG bit turned on in the TCP header. (client→server)
76	urgent_data_pkts_*b a*	” (server→client)
77	urgent_data_bytes_*a b*	The total bytes of urgent data sent. This field is calculated by summing the urgent pointer offset values found in packets having the URG bit set in the TCP header. (client→server)
78	urgent_data_bytes_*b a*	” (server→client)
79	mss_requested_*a b*	The Maximum Segment Size (MSS) requested as a TCP option in the SYN packet opening the connection. (client→server)
80	mss_requested_*b a*	” (server→client)
81	max_segm_size_*a b*	The maximum segment size observed during the lifetime of the connection. (client→server)
82	max_segm_size_*b a*	” (server→client)
83	min_segm_size_*a b*	The minimum segment size observed during the lifetime of the connection. (client→server)
84	min_segm_size_*b a*	” (server→client)
85	avg_segm_size_*a b*	The average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data pkts reported. (client→server)
86	avg_segm_size_*b a*	” (server→client)
87	max_win_adv_*a b*	The maximum window advertisement seen. If the connection is using window scaling (both sides negotiated window scaling during the opening of the connection), this is the maximum window-scaled advertisement seen in the connection. For a connection using window scaling, both the SYN segments opening the connection have to be captured in the dumpfile for this and the following window statistics to be accurate. (client→server)
88	max_win_adv_*b a*	” (server→client)
89	min_win_adv_*a b*	The minimum window advertisement seen. This is the minimum window-scaled advertisement seen if both sides negotiated window scaling. (client→server)
90	min_win_adv_*b a*	” (server→client)
91	zero_win_adv_*a b*	The number of times a zero receive window was advertised. (client→server)
92	zero_win_adv_*b a*	” (server→client)
93	avg_win_adv_*a b*	The average window advertisement seen, calculated as the sum of all window advertisements divided by the total number of packets seen. If the connection endpoints negotiated window scaling, this average is calculated as the sum of all window-scaled advertisements divided by the number of window-scaled packets seen. Note that in the window-scaled case, the window advertisements in the SYN packets are excluded since the SYN packets themselves cannot have their window advertisements scaled, as per RFC 1323 [8]. (client→server)

Continued on next page

Number	Short	Long
94	avg_win_adv_*b a*	” (server→client)
95	initial_window-bytes_*a b*	The total number of bytes sent in the initial window i.e., the number of bytes seen in the initial flight of data before receiving the first ack packet from the other endpoint. Note that the ack packet from the other endpoint is the first ack acknowledging some data (the ACKs part of the 3-way handshake do not count), and any retransmitted packets in this stage are excluded. (client→server)
96	initial_window-bytes_*b a*	” (server→client)
97	initial_window-packets_*a b*	The total number of segments (packets) sent in the initial window as explained above. (client→server)
98	initial_window-packets_*b a*	” (server→client)
99	ttl_stream_length_*a b*	The Theoretical Stream Length. This is calculated as the difference between the sequence numbers of the SYN and FIN packets, giving the length of the data stream seen. Note that this calculation is aware of sequence space wrap-arounds, and is printed only if the connection was complete (both the SYN and FIN packets were seen). (client→server)
100	ttl_stream_length_*b a*	” (server→client)
101	missed_data_*a b*	The missed data, calculated as the difference between the ttl stream length and unique bytes sent. If the connection was not complete, this calculation is invalid and an “NA” (Not Available) is printed. (client→server)
102	missed_data_*b a*	” (server→client)
103	truncated_data_*a b*	The truncated data, calculated as the total bytes of data truncated during packet capture. For example, with tcpdump, the snaplen option can be set to 64 (with -s option) so that just the headers of the packet (assuming there are no options) are captured, truncating most of the packet data. In an Ethernet with maximum segment size of 1500 bytes, this would amount to truncated data of 1500 64 = 1436bytes for a packet. (client→server)
104	truncated_data_*b a*	” (server→client)
105	truncated_packets_*a b*	The total number of packets truncated as explained above. (client→server)
106	truncated_packets_*b a*	” (server→client)
107	data_xmit_time_*a b*	Total data transmit time, calculated as the difference between the times of capture of the first and last packets carrying non-zero TCP data payload. (client→server)

Continued on next page

Number	Short	Long
108	data_xmit_time_*b a*	” (server→client)
109	idletime_max_*a b*	Maximum idle time, calculated as the maximum time between consecutive packets seen in the direction. (client→server)
110	idletime_max_*b a*	” (server→client)
111	throughput_*a b*	The average throughput calculated as the unique bytes sent divided by the elapsed time i.e., the value reported in the unique bytes sent field divided by the elapsed time (the time difference between the capture of the first and last packets in the direction). (client→server)
112	throughput_*b a*	” (server→client)
113	RTT_samples_*a b*	The total number of Round-Trip Time (RTT) samples found. tcptrace is pretty smart about choosing only valid RTT samples. An RTT sample is found only if an ack packet is received from the other endpoint for a previously transmitted packet such that the acknowledgment value is 1 greater than the last sequence number of the packet. Further, it is required that the packet being acknowledged was not retransmitted, and that no packets that came before it in the sequence space were retransmitted after the packet was transmitted. Note : The former condition invalidates RTT samples due to the retransmission ambiguity problem, and the latter condition invalidates RTT samples since it could be the case that the ack packet could be cumulatively acknowledging the retransmitted packet, and not necessarily ack-ing the packet in question. (client→server)
114	RTT_samples_*b a*	” (server→client)
115	RTT_min_*a b*	The minimum RTT sample seen. (client→server)
116	RTT_min_*b a*	” (server→client)
117	RTT_max_*a b*	The maximum RTT sample seen. (client→server)
118	RTT_max_*b a*	” (server→client)
119	RTT_avg_*a b*	The average value of RTT found, calculated straightforward-ly as the sum of all the RTT values found divided by the total number of RTT samples. (client→server)
120	RTT_avg_*b a*	” (server→client)
121	RTT_stdv_*a b*	The standard deviation of the RTT samples. (client→server)
122	RTT_stdv_*b a*	” (server→client)
123	RTT_from_3WHS_*a b*	The RTT value calculated from the TCP 3-Way Hand-Shake (connection opening) [9], assuming that the SYN packets of the connection were captured. (client→server)

Continued on next page

Number	Short	Long
124	RTT_from_3WHS_*b a*	" (server→client)
125	RTT_full_sz_smpls_*a b*	The total number of full-size RTT samples, calculated from the RTT samples of full-size segments. Full-size segments are defined to be the segments of the largest size seen in the connection. (client→server)
126	RTT_full_sz_smpls_*b a*	" (server→client)
127	RTT_full_sz_min_*a b*	The minimum full-size RTT sample. (client→server)
128	RTT_full_sz_min_*b a*	" (server→client)
129	RTT_full_sz_max_*a b*	The maximum full-size RTT sample. (client→server)
130	RTT_full_sz_max_*b a*	" (server→client)
131	RTT_full_sz_avg_*a b*	The average full-size RTT sample. (client→server)
132	RTT_full_sz_avg_*b a*	" (server→client)
133	RTT_full_sz_stdev_*a b*	The standard deviation of full-size RTT samples. (client→server)
134	RTT_full_sz_stdev_*b a*	" (server→client)
135	post-loss_acks_*a b*	The total number of ack packets received after losses were detected and a retransmission occurred. More precisely, a post-loss ack is found to occur when an ack packet acknowledges a packet sent (acknowledgment value in the ack pkt is 1 greater than the packet's last sequence number), and at least one packet occurring before the packet acknowledged, was retransmitted later. In other words, the ack packet is received after we observed a (perceived) loss event and are recovering from it. (client→server)
136	post-loss_acks_*b a*	" (server→client)
137	segs_cum_acked_*a b*	The count of the number of segments that were cumulatively acknowledged and not directly acknowledged. (client→server)
138	segs_cum_acked_*b a*	" (server→client)
139	duplicate_acks_*a b*	The total number of duplicate acknowledgments received. (client→server)
140	duplicate_acks_*b a*	" (server→client)
141	triple_dupacks_*a b*	The total number of triple duplicate acknowledgments received (three duplicate acknowledgments acknowledging the same segment), a condition commonly used to trigger the fast-retransmit/fast-recovery phase of TCP. (client→server)
142	triple_dupacks_*b a*	" (server→client)
143	max_#_retrans_*a b*	The maximum number of retransmissions seen for any segment during the lifetime of the connection. (client→server)

Continued on next page

Number	Short	Long
144	max_#_retrans_*b a*	” (server→client)
145	min_retr_time_*a b*	The minimum time seen between any two (re)transmissions of a segment amongst all the retransmissions seen. (client→server)
146	min_retr_time_*b a*	” (server→client)
147	max_retr_time_*a b*	The maximum time seen between any two (re)transmissions of a segment. (client→server)
148	max_retr_time_*b a*	” (server→client)
149	avg_retr_time_*a b*	The average time seen between any two (re)transmissions of a segment calculated from all the retransmissions. (client→server)
150	avg_retr_time_*b a*	” (server→client)
151	sdv_retr_time_*a b*	The standard deviation of the retransmission-time samples obtained from all the retransmissions. (client→server)
152	sdv_retr_time_*b a*	” (server→client)
153	min_data_wire_*a b*	Minimum number of bytes in (Ethernet) packet (client→server)
154	q1_data_wire_*a b*	First quartile of bytes in (Ethernet) packet
155	med_data_wire_*a b*	Median of bytes in (Ethernet) packet
156	mean_data_wire_*a b*	Mean of bytes in (Ethernet) packet
157	q3_data_wire_*a b*	Third quartile of bytes in (Ethernet) packet
158	max_data_wire_*a b*	Maximum of bytes in (Ethernet) packet
159	var_data_wire_*a b*	Variance of bytes in (Ethernet) packet
160	min_data_ip_*a b*	Minimum number of total bytes in IP packet
161	q1_data_ip_*a b*	First quartile of total bytes in IP packet
162	med_data_ip_*a b*	Median of total bytes in IP packet
163	mean_data_ip_*a b*	Mean of total bytes in IP packet
164	q3_data_ip_*a b*	Third quartile of total bytes in IP packet
165	max_data_ip_*a b*	Maximum of total bytes in IP packet
166	var_data_ip_*a b*	Variance of total bytes in IP packet
167	min_data_control_*a b*	Minimum of control bytes in packet
168	q1_data_control_*a b*	First quartile of control bytes in packet
169	med_data_control_*a b*	Median of control bytes in packet
170	mean_data_control_*a b*	Mean of control bytes in packet
171	q3_data_control_*a b*	Third quartile of control bytes in packet
172	max_data_control_*a b*	Maximum of control bytes in packet
173	var_data_control_*a b*	Variance of control bytes packet
174	min_data_wire_*b a*	Minimum number of bytes in (Ethernet) packet (server→client)
175	q1_data_wire_*b a*	First quartile of bytes in (Ethernet) packet
176	med_data_wire_*b a*	Median of bytes in (Ethernet) packet
177	mean_data_wire_*b a*	Mean of bytes in (Ethernet) packet

Continued on next page

Number	Short	Long
178	q3_data_wire_*b a*	Third quartile of bytes in (Ethernet) packet
179	max_data_wire_*b a*	Maximum of bytes in (Ethernet) packet
180	var_data_wire_*b a*	Variance of bytes in (Ethernet) packet
181	min_data_ip_*b a*	Minimum number of total bytes in IP packet
182	q1_data_ip_*b a*	First quartile of total bytes in IP packet
183	med_data_ip_*b a*	Median of total bytes in IP packet
184	mean_data_ip_*b a*	Mean of total bytes in IP packet
185	q3_data_ip_*b a*	Third quartile of total bytes in IP packet
186	max_data_ip_*b a*	Maximum of total bytes in IP packet
187	var_data_ip_*b a*	Variance of total bytes in IP packet
188	min_data_control_*b a*	Minimum of control bytes in packet
189	q1_data_control_*b a*	First quartile of control bytes in packet
190	med_data_control_*b a*	Median of control bytes in packet
191	mean_data_control_*b a*	Mean of control bytes in packet
192	q3_data_control_*b a*	Third quartile of control bytes in packet
193	max_data_control_*b a*	Maximum of control bytes in packet
194	var_data_control_*b a*	Variance of control bytes packet
195	min_IAT_*a b*	Minimum of packet inter-arrival time (client→server)
196	q1_IAT_*a b*	First quartile of packet inter-arrival time
197	med_IAT_*a b*	Median of packet inter-arrival time
198	mean_IAT_*a b*	Mean of packet inter-arrival time
199	q3_IAT_*a b*	Third quartile of packet inter-arrival time
200	max_IAT_*a b*	Maximum of packet inter-arrival time
201	var_IAT_*a b*	Variance of packet inter-arrival time
202	min_IAT_*b a*	Minimum of packet inter-arrival time (server→client)
203	q1_IAT_*b a*	First quartile of packet inter-arrival time
204	med_IAT_*b a*	Median of packet inter-arrival time
205	mean_IAT_*b a*	Mean of packet inter-arrival time
206	q3_IAT_*b a*	Third quartile of packet inter-arrival time
207	max_IAT_*b a*	Maximum of packet inter-arrival time
208	var_IAT_*b a*	Variance of packet inter-arrival time
209	Time_since_last_connection	Time since the last connection between these hosts
210	No._transitions_bulk/trans	The number of transitions between transaction mode and bulk transfer mode, where bulk transfer mode is defined as the time when there are more than three successive packets in the same direction without any packets carrying data in the other direction
211	Time_spent_in_bulk	Amount of time spent in bulk transfer mode
212	Duration	Connection duration
213	%_bulk	Percent of time spent in bulk transfer
214	Time_spent_idle	The time spent idle (where idle time is the accumulation of all periods of 2 seconds or greater when no packet was seen in either direction)

Continued on next page

Number	Short	Long
215	%_idle	Percent of time spent idle
216	Effective_Bandwidth	Effective Bandwidth based upon entropy [10] (both directions)
217	Effective_Bandwidth_*a b*	” (client→server)
218	Effective_Bandwidth_*b a*	” (server→client)
219	FFT_all	FFT of packet IAT (arctan of the top-ten frequencies ranked by the magnitude of their contribution) (all traffic) (Frequency #1)
220	FFT_all	” (Frequency #2)
221	FFT_all	” ...
222	FFT_all	” ...
223	FFT_all	” ...
224	FFT_all	” ...
225	FFT_all	” ...
226	FFT_all	” ...
227	FFT_all	” ...
228	FFT_all	” (Frequency #10)
229	FFT_*a b*	FFT of packet IAT (arctan of the top-ten frequencies ranked by the magnitude of their contribution) (client→server) (Frequency #1)
230	FFT_*a b*	” (Frequency #2)
231	FFT_*a b*	” ...
232	FFT_*a b*	” ...
233	FFT_*a b*	” ...
234	FFT_*a b*	” ...
235	FFT_*a b*	” ...
236	FFT_*a b*	” ...
237	FFT_*a b*	” ...
238	FFT_*b a*	” (Frequency #10)
239	FFT_*b a*	FFT of packet IAT (arctan of the top-ten frequencies ranked by the magnitude of their contribution) (server→client) (Frequency #1)
240	FFT_*b a*	” (Frequency #2)
241	FFT_*b a*	” ...
242	FFT_*b a*	” ...
243	FFT_*b a*	” ...
244	FFT_*b a*	” ...
245	FFT_*b a*	” ...
246	FFT_*b a*	” ...
247	FFT_*b a*	” ...
248	FFT_*b a*	” (Frequency #10)
249	Classes	Application class, as assigned in [1]

Appendix C: MATLAB Code for Neural Network Processing

The following code was generated from the MATLAB neural network tool, and then modified by CPT James Jablonski and Maj Kristy Moore for use with this thesis research.

```
function [targets, netouts, removed,overalltrue ,cmconfuse, timevar] =
netsnr1outputtf_Moore1( Data,attempts,doSNR,usegpu, trainmodeinput )

%This function creates a Neural Network and performs SNR on a given data set
% The Data must include rows of exemplars and columns of features.
% This function Assumes data labels are in column 1 and parses them into the proper
form for NN training
%
%Data = Data inputs
%num attempts = Number of iterations of training (will return the net with the least
cross-entropy)
%doSNR = 0 don't perform SNR on data end on full feature set. else=DO
%USEGPU = 0 don't use GPU.  1= use gpu with PURELIN transfer function.
%train = type of training for the neural net; use 1 for trainscg, 2 for
%trainbr, 3 traingdm, 4 traingda, 5 traincgp, 6 traingcgf, 7 trainbfg, %8traincgb, 9
traingdx, 10 trainlm, 11 trainoss, or 12 trainrp (this %could change to a for loop that
cycles through these in the future)

%Based on the function input, choose which training algorithm to use
trainingmodes = {'trainscg';'trainbr';'traingdm';'traingda';'traincgp';'traincgf';...
'trainbfg';'traincgb';'traingdx'; 'trainlm';'trainoss';'traingrp'};

%trainingmodes = cellstr(trainingmodelist);
trainmodefunc = trainingmodes(trainmodeinput);

%the number of hidden nodes in the structure
numhidden = [10 20 30 40 50];

for b =1:length(numhidden) %Loop through running with a different node structure each
time

    %start the stopwatch to track how long the processing takes
    tic

    %create the save name for this run
    trainmodename = char(trainmodefunc);
    savename = strcat(inputname(1),'_',num2str(numhidden(b)), ...
        '_',num2str(attempts),'_',trainmodename);
    mkdir(savename);
    currentfolder = pwd;
```

```
savepath = fullfile(currentfolder,savename);

%count the number of classes
classvalues = unique(Data(:,1));
numclasses = length(classvalues);

%See if 0 is a class
if any(classvalues)==0
   n = 0;
else
   n = 1;
end

%allocate size for datakey vector
%if there are 2 classes, say (0,1) then we only need 1 datakey %column
if numclasses == 2 && n == 0
   datakeys = zeros(length(Data),1);
else
   datakeys = zeros(length(Data),numclasses);
end

for i=1:length(Data(:,1)) %go through the first column
   %check for 0,1 class which can be considered 1 class
   if numclasses == 2 && n == 0
      if Data(i,1) ~= 0
         datakeys(i,1) = 1; %for whatever the other class value is
      end
   else
      for j=1:numclasses %cycle through the class values
         if Data(i,1)==classvalues(j)
            if n == 0
               datakeys(i,j+1)=1; %need a column for the zero values
            else
               datakeys(i,j)=1; %need columns only for class values
            end
         end
      end
   end
end

%chop off first column (class)
data = Data(:,2:size(Data,2));

%standardize my data
meanV = repmat(mean(data),length(data),1); %feature means vector
```

```
stdev = repmat(std(data),length(data),1);  %prepare vector of %feature stdev

%sub mean to center
data = data - meanV;

%divide by std's
data = data./stdev;

%add noise vector
data = [rand(length(data),1),data];

%set up the neural nets input and output
input = data;
output = datakeys;

% "remember" removed feature (for SNR)- bookkeeping
featuresremaining = 1:size(data,2);

snrs = zeros(size(data,2),size(data,2)); %a place to store all %SNR's

%Loop through creating several nets, then pick the best one
%then select fewer features using SNR

for z = 1:(size(data,2)-1);   %iterate through all features until 2 %are left (includes
   noise)
    nets = {};    %create the cell array to store the nets
    perfs = [];   %create the array to store the performance values

  %if not doing the SNR feature removal then this loop only happens once
    if doSNR == 0
        z = size(data,2)-1;
    end

    %neural network requires them to be transposed
    inputs = input';
    targets = output';

    %set up the network
    net = patternnet(numhidden(b));

    %Loop through all desired attempts at creating nets
    for k = 1:attempts

        %Initialize network weights and biases after 1st run (initialized on net creation)
        if k > 1
```

```
    net = init(net);
end

% Pre/Post functions
net.inputs{1}.processFcns = {'mapminmax','removeconstantrows'};
          %,'removeconstantrows'
net.outputs{1}.processFcns = {'removeconstantrows'};
          %'removeconstantrows'

%Specifiy 'logsig' or 'purelin'  the transfer function at each layer
if usegpu==1
    net.layers{1}.transferFcn = 'purelin';
    net.layers{2}.transferFcn = 'purelin';
else
    net.layers{1}.transferFcn = 'logsig';
    net.layers{2}.transferFcn = 'logsig';
end

% Data manipulations
net.divideFcn = 'dividerand';  %divide up the data randomly
net.divideMode = 'sample';  % Divide by sample
net.divideParam.trainRatio = 70/100; %70% for training
net.divideParam.valRatio = 15/100; %15% for validation
net.divideParam.testRatio = 15/100; %15% for testing

%  help nntrain
net.trainFcn = trainmodename;  % Selected training mode from above
net.trainParam.epochs = 500; % Specify training epochs
net.trainParam.time = 300;   % Specify max training time
net.trainParam.goal = .005; % Specify training error goal def .005
net.trainParam.showWindow = 1;  % 0 = Don't show the training GUI
net.trainParam.max_fail = 10; %number of validation failures

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'crossentropy';  % Cross-Entropy

% train
if usegpu==1
    [net,tr] = train(net,inputs,targets,'useGPU','yes');
else
    [net,tr] = train(net,inputs,targets);
end

%run it
```

```
    outputs = net(inputs);

    %get performance data from the network created
    errors = gsubtract(targets,outputs);
    performance = perform(net,targets,outputs);
    perfs(k,1)=performance; %record crossentropys
    nets{k,1}=net;

end

[v,I]=min(perfs); %find index of best net by crossentropy

perfsout(z) = v;

outputs = nets{I,1}(inputs); % get outputs from best net
NetOutputs{z} = outputs; %track the best outputs
netout=nets{I,1};
netouts{z}=netout;

%Get the confusion matrix information and store it
%Am not currently using the ind - can change later if needed
[c,cm,ind,per] = confusion(targets, outputs);
perfconfuse{z} = per;
cmconfuse{z} = cm;

%Overall classification accuracy
overalltrue(z)=1-c

%Get the ROC Curve data with posclass for each class level
[tpr, fpr, thresholds] = roc(targets, outputs);
tprout{z} = tpr;
fprout{z} = fpr;
thresholdsout{z} = thresholds;
rocInfo{z} = {tprout{z}, fprout{z}, thresholdsout{z}};

%convert the iteration number to a string
num = num2str(z);

%plot the ROC curves
figure(z), plotroc(targets,outputs);
%create the save name for the ROC curves
ROC_curves_savename = strcat('ROCCurve_',savename,'(',num,').fig');
%save the figure then close it
savefig(fullfile(savepath,ROC_curves_savename));
close(gcf)
```

```
%plot the performance
figure(z), plotperform(tr);
%create the save name for the performance curves
perform_curves_savename = strcat('PerformCurve_',savename,'(',num,').fig');
%save the figure then close it
savefig(fullfile(savepath,perform_curves_savename));
close(gcf)

%plot the training state
figure(z), plottrainstate(tr);
%create the save name for the train state plot
trainstate_savename = strcat('TrainState_',savename,'(',num,').fig');
%save the figure then close it
savefig(fullfile(savepath,trainstate_savename));
close(gcf)

%plot the error history
figure(z), ploterrhist(errors,'bins',20);
%create the save name for the error history
ErrHistory_savename = strcat('ErrHistory_',savename,'(',num,').fig');
%save the figure then close it
savefig(fullfile(savepath,ErrHistory_savename));
close(gcf)

%plot the final confusion matrix
figure(z), plotconfusion(targets,outputs);
%create the save name for the figure
confusion_plot_savename = strcat('ConfusionPlot_',savename,'(',num,').fig');
%save the figure then close it
savefig(fullfile(savepath,confusion_plot_savename));
close(gcf)

if doSNR == 1
   %do the SNR check
   snr = [];
   wts = net.IW{1,1}; %create the weights for SNR
   dim = size(wts);
   noise = wts(:,1)'*wts(:,1);
   for j = 2:dim(2) %calculate the SNR values
      snr(j)=10*log10((wts(:,j)'*wts(:,j))/noise);
   end

   %create the array for bookkeeping
   row = zeros(1,size(snrs,2));
```

```
        %adjust the array of features remaining
        for d = 1:length(snr)
            index = featuresremaining(d);
            row(1,index) = row(1,index)+snr(d);
        end

        snrs(z,:) = row;

        %plot the features remaining vs. their snrs
        figure(z), bar(featuresremaining,snr);
        bar_plot_savename = strcat('SNR_BarPlot_',savename,'(',num,').fig');
        %save the figure then close it
        savefig(fullfile(savepath,bar_plot_savename));
        close(gcf)

        snr(1)=100; %make sure I don't remove noise

        %Remove Least significant Features in Order of SNR loop %through trials again

        [val, I]=min(snr); %index of smallest SNR = I

        %remove index of smallest feature from featuresremaining
        removed(z)=featuresremaining(I);

        %check to see if end of array
        if I==size(featuresremaining,2)
            featuresremaining=featuresremaining(:,1:I-1);
        else
            featuresremaining=[featuresremaining(:,1:I-
                        1),featuresremaining(:,I+1:dim(2))];
        end

        %remove smallest feature; check to see if end of array
        if I==dim(2)
            input=input(:,1:I-1);
        else
            input=[input(:,1:I-1),input(:,I+1:dim(2))];
        end
    end
end

%stop the timer
timevar = toc;

%save the workspace and its associated variables
```

```
    saveworkspace = strcat(savename, '.mat');
    save(saveworkspace)

end

end
```

References

[1] H. Katzan Jr., "Essentials of Cybersecurity," in *Southeastern INFORMS Conference*, Myrtle Beach, 2012.

[2] October 2013. [Online]. Available: http://www.militarycybersecurity.com/agenda-military-cyber-security-conference/. [Accessed 10 January 2014].

[3] "The Economic Impact of Cybercrime and Cyber Espionage," 2013.

[4] "Department of Homeland Security Strategic Plan Fiscal Years 2012-2016," 2012.

[5] S. Ackerman, "Cyber-Attacks Eclipsing Terrorism as Gravest Domestic Threat," *The Guardian,* November 2013.

[6] A. W. Moore, D. Zuev and M. L. Crogran, "Discrimators for use in flow-based classification," Department of Computer Science, Queen Mary University of London, London, 2005.

[7] C. Janssen, 2013. [Online]. Available: http://www.techopedia.com/. [Accessed 05 November 2013].

[8] M. Elizabeth, October 2013. [Online]. Available: http://www.wisegeek.com/what-is-a-computer-audit.htm. [Accessed 5 November 2013].

[9] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 6 ed., Pearson Education, 2012.

[10] C. M. Kozierok, September 2005. [Online]. Available: http://tech911.info/html/body_it_training_help.html. [Accessed 05 November 2013].

[11] "Oxford Dictionary," October 2013. [Online]. Available: http://oxforddictionaries.com/us/definition/american_english/server. [Accessed 05 November 2013].

[12] D. Chandrasekaran, August 2002. [Online]. Available: http://www.boloji.com/index.cfm?md=Content&sd=Articles&ArticleID=497. [Accessed 05 November 2013].

[13] "ITT Training," October 2013. [Online]. Available: http://tech911.info/html/body_it_training_help.html. [Accessed 5 November 2013].

[14] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys and Tutorials,* vol. 10, no. 4, pp. 56 - 76, 2008.

[15] A. Dainotti, A. Pescape and K. C. Claffy, "Issues and Future Directions in Traffic Classification," *IEEE Network,* vol. 26, no. 1, pp. 35-40, January/February 2012.

[16] A. W. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," *Proeedings of the 6th Passive Active Measurement (PAM) Workshop,* vol. 3431, pp. 41-54, March 2005.

[17] "IANA Port Numbers," January 2014. [Online]. Available: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. [Accessed 15 January 2014].

[18] M. Roughan, S. Sen, O. Spatscheck and N. Duffield, "Class-of-Service Mapping for QoS: A statistical signature-based approach to IP traffic classification," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Sicily, 2004.

[19] P. Haffner, S. Sen, O. Spatscheck and D. Wang, "ACAS: Automated Construction of Application Signatures," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining network data*, Philadelphia, 2005.

[20] A. K. Jain and J. Mao, "Artificial Neural Networks: A Tutorial," *IEEE Computer,* vol. 29, no. 3, pp. 31-44, March 1996.

[21] A. McGregor, M. Hall, P. Lorier and J. Brunskill, "Flow Clustering Using Machine Learning Techniques," in *Proceedings of the 5th Passive and Active Measurement Workshop (PAM 2004)*, Berlin Heidelberg, 2004.

[22] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," *ACM SIGMETRICS Performance Evaluation Review,* vol. 33, no. 1, pp. 50-60, 2005.

[23] S. Zander, T. Nguyen and G. Armitage, "Automated traffic classification and application identification using maching learning," in *30th Annual IEEE Conference on Local Computer Networks*, Sydbey, 2005.

[24] J. Erman, M. Arlitt and A. Mahanti, "Traffic classification using clustering algorithms," in *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, ACM, 2006.

[25] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review,* vol. 36, no. 2, pp. 23 - 26, April 2006.

[26] "UC Davis Statwiki," January 2014. [Online]. Available: http://statwiki.ucdavis.edu/Nonparametric_Inference/Kernel_Density_Estimation. [Accessed 15 January 2014].

[27] N. Williams, S. Zander and G. Armitage, "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification," *ACM SIGCOMM Computer Communication Review,* vol. 36, no. 5, pp. 5 - 16, October 2006.

[28] J. Zhang, C. Chen, W. Zhou and Y. Xiang, "Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions," *IEEE Transactions on Information Forensics and Security,* vol. 8, no. 1, pp. 5 - 15, January 2013.

[29] J. Erman, A. Mahanti, M. Arlitt, I. Cohen and C. Williamson, "Offline/Realtime Classification Using Semi-Supervised Learning," *Performance Evaluation,* vol. 64, no. 9 , pp. 1194 - 1213, October 2007.

[30] S. Huang, K. Chen, C. Liu, A. Liang and H. Guan, "A Statistical-Feature-Based Approach to Internet Traffic Classification Using Machine Learning," in *International Conference on Ultra Modern Telecommunications Workshops (ICUMT '09)*, 2009.

[31] T. Auld, A. W. Moore and S. F. Gull, "Bayesian Neural Networks for Internet Classification," *IEEE TRANSACTIONS ON NEURAL NETWORKS,* vol. 18, no. 1, pp. 223-239, January 2007.

[32] W. Zhou, L. Dong, L. Bic, M. Zhou and L. Chen, "Internet Classification Using Feed-Forward Neural Network," in *2011 International Conference on Computational Problem-Solving (ICCP)*, Chengdu, 2011.

[33] W. Li and A. W. Moore, "A machine learning approach for efficient traffic classification," in *15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS '07*, Istanbul, 2007.

[34] T. S. Tabatabaei, F. Karray and M. Kamel, "Early Internet Traffic Recognition Based on Machine Learning Methods," in *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Montreal, 2012.

[35] Y. Zhang, Y. Zhou and K. Chen, "Internet Traffic Classification based on Bag-of-Words Model," in *2012 IEEE Globecom Workshops (GC Wkshps)*, Anaheim, 2012.

[36] M. Crotti, M. Dusi, F. Gringoli and L. Salgarelli, "Traffic Classification through Simple Statistical Fingerprinting," *ACM SIGCOMM Computer Communication Review,* vol. 37, no. 1, pp. 5 - 16, 2007.

[37] L. Li and K. Kianmehr, "Internet Traffic Classification based on Associative Classifiers," in *2012 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems (CYBER)*, Bangkok, 2012.

[38] M. Panda and M. R. Patra, "Network Intrusion Detection Using Naive Bayes," *International Journal of Computer Science and Network Security,* vol. 7, no. 12, pp. 258 - 263, December 2007.

[39] L. Portnoy, E. Eskin and S. Stolfo, "Intrusion Detection with Unlabeled Data Using Clustering," in *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, 2001.

[40] S. Zanero and S. M. Savaresi, "Unsupervised Learning Techniques for an Intrusion Detection System," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, New York, 2004.

[41] Z. S. Pan, S. C. Chen, G. B. Hu and D. Q. Zhang, "Hybrid Neural Network and C4.5 for Misuse Detection," in *Proceedings of the 2nd International Conference on Machine Learning and Cybernetics*, Xi'an, 2003.

[42] M. Moradi and M. Zulkernine, "A Neural Network Based System for Intrusion Detection and Classification of Attacks," in *Proceedings of 2004 IEEE International Conference on Advances in Intelligent Systems Theory and Applications*, Luxembourg Kirchberg, 2004.

[43] X. Xu and X. Wang, "An Adaptive Network Intrusion Detection Method Based on PCA and Support Vector Machines," in *Advanced Data Mining and Applications*, X. Li, S. Wang and Z. Y. Dong, Eds., Wuhan, Springer Berlin Heidelberg, 2005, pp. 696 - 703.

[44] W. Hu, W. Hu and S. Maybank, "AdaBoost-Based Algorithm for Network Intrusion Detection," *IEEE Transactions on Systems, Man, and Cybernetics -- Part B, Cybernetics,* vol. 38, no. 2, pp. 577 - 583, April 2008.

[45] O. Linda, T. Vollmer and M. Mainc, "Neural Network Based Intrusion Detection System for Critical Infrastructures," in *IEEE International Joint Conference on Neural Networks*, Atlanta, 2009.

[46] M. V. Mahoney and P. K. Chan, "Learning Rules for Anomaly Detection of Hostile Network Traffic," in *Third IEEE International Conference on Data Mining (ICDM)*, Melbourne, 2003.

[47] G. Stein, B. Chen, A. S. Wu and K. A. Hua, "Decision Tree Classifier for Network Intrusion Detection with GA-Based Feature Selection," in *Proceedings of the 43rd Southeast Regional Conference (ACM) - Volume 2*, Kennesaw, 2005.

[48] O. Linda, M. Manic, T. Vollmer and J. Wright, "Fuzzy Logic Based Anomaly Detection for Embedded Network Security Cyber Sensor," in *IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, Paris, 2011.

[49] M. Faloutsos, "Detecting Malware with Graph-based Methods: Traffic Classification, Botnets, and Facebook Scams," in *Proceedings of the 22nd International Conference on World Wide Web Companion*, Rio de Janeiro, 2013.

[50] R. Setiono and H. Liu, "Neural-Network Feature Selector," *IEEE Transactions on Neural Networks,* vol. 8, no. 3, pp. 654 - 662, May 1997.

[51] L. M. Belue and K. W. Bauer Jr., "Determining Input Features for Multilayer Perceptrons," *Neurocomputing,* vol. 7, no. 2, pp. 111 - 121, March 1995.

[52] J. M. Steppe and K. W. Bauer Jr., "Improved Feature Screening in Feedforward Neural Networks," *Neurocomputing,* vol. 13, no. 1, pp. 47 - 58, September 1996.

[53] A. Verikas and M. Bacauskiene, "Feature Selection with Neural Networks," *Pattern Recognition Letters,* vol. 23, no. 11, pp. 1323 - 1335, September 2002.

[54] K. W. Bauer Jr., S. G. Alsing and K. A. Greene, "Feature Screening Using Signal-to-Noise Ratios," *Neurocomputing,* vol. 31, no. 1-4, pp. 29-44, March 2000.

[55] B. E. Mullins, T. H. Lacey, R. F. Mills, J. M. Trechter and S. D. Bass, "How the Cyber Defense Exercise Shaped an Information-Assurance Curriculum," *IEEE Security & Privacy,* vol. 5, no. 5, pp. 40-49, October 2007.

[56] "TCPDump," [Online]. Available: http://www.tcpdump.org. [Accessed 15 November 2013].

[57] "Wireshark," 2013. [Online]. Available: http://www.wireshark.org. [Accessed 30 December 2013].

[58] "BRASIL Downloads," 2009. [Online]. Available: http://www.cl.cam.ac.uk/research/srg/netos/brasil/downloads/. [Accessed 15 November 2013].

[59] J. W. Ji, "Holistic Network Defense: Fusing Host and Network Features for Attack Classification," Wright-Patterson AFB, OH, 2001.

[60] D. Burks, 2013. [Online]. Available: http://blog.securityonion.net/. [Accessed 15 November 2013].

[61] A. Turner, 2013. [Online]. Available: https://github.com/synfinatic/tcpreplay. [Accessed 15 November 2013].

[62] B. Visscher, 2007. [Online]. Available: http://sguil.sourceforge.net/. [Accessed 15 November 2013].

[63] "Snort," 2013. [Online]. Available: http://www.snort.org/. [Accessed 04 January 2014].

[64] "WEKA," 2013. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/index.html. [Accessed 15 November 2013].

[65] "Microsoft Support," September 2011. [Online]. Available: http://support.microsoft.com/kb/828795. [Accessed 15 January 2014].

[66] K. W. Bauer, "OPER 685 Multivariate Analysis - Class Notes," Wright-Patterson AFB, 2013.

[67] V. Cheung and K. Cannons, "An Introduction to Neural Networks," Winnpeg, 2002.

[68] D. E. Rumelhart, G. E. Hintont and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature,* vol. 323, no. 9, pp. 533-536, October 1986.

[69] J. Steppe and K. Bauer Jr., "Feature Saliency Measures," *Computers & Mathematics with Applications,* vol. 33, no. 8, pp. 109 - 126, April 1997.

[70] G. L. Tarr, "Multi-layered Feedforward Neural Networks for Image Segmentation," Wright-Patterson AFB, OH, 1991.

[71] "Mathworks," 2013. [Online]. Available: http://www.mathworks.com/products/neural-network/. [Accessed 30 December 2013].

[72] V. Labatut and H. Cherifi, "Accuracy Measures for the Comparison of Classifiers," in *arXiv: 1207.3790*, Amman, 2012.

[73] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation," *Journal of Machine Learning Technologies,* vol. 2, no. 1, pp. 37-63, February 2011.

[74] M. Barjaktarovic, May 2013. [Online]. Available: http://www.cs.odu.edu/~mukka/cs795sum13dm/Lecturenotes/Day4/recallprecision.pdf. [Accessed 04 January 2013].

[75] A. Slaby, "ROC Analysis with Matlab," in *29th International Conference on Information Technology Interfaces*, Cavtat, Croatia, 2007.

[76] J. D. McCaffrey, November 2013. [Online]. Available: http://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/. [Accessed 30 December 2013].

Vita

Major Kristy L. Moore is currently a graduate student pursuing a degree in Operations Research at the Air Force Institute of Technology. She graduated from Dominican University of California with a Bachelor of Arts Degree in Mathematics in 2001. She received her commission from Officer Training School at Maxwell Air Force Base, Alabama. Maj Moore also has a Master's of Computer Information Systems from the University of Phoenix and is CompTIA A+ certified.

Her previous assignments include Deputy Program Manager, Air Force Research Laboratory Human Effectiveness Bioacoustics Branch; Directed Energy Weapons Modeler, National Air and Space Intelligence Center; Chief of Analysis, F-15C Division, 59th Test and Evaluation Squadron; and Project Manager, National Assessment Group. She has also been a member of the Department of Defense Space Countermeasures Hands-On Program and served in direct support of OPERATION NOBLE EAGLE. After graduation Major Moore will be assigned to Detachment 1, 609th Air Operations Center, Shaw Air Force Base, South Carolina.

SF298

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704–0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD–MM–YYYY)	2. REPORT TYPE	3. DATES COVERED *(From — To)*
27-03-2014	Master's Thesis	Sep 2012 – Mar 2014

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Salient feature selection using feed-forward neural networks and signal-to-noise ratios with a focus toward network threat detection and risk level identification	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Moore, Kristy L, Major, USAF	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENY) 2950 Hobson Way WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENS-14-M-22
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RYA POC: Lt Col David Ryer, David.Ryer@us.af.mil 2241 Avionics Circle Area B, Building 620 WPAFB, OH 45433-7321 (937)528-8389	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED	
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.	

14. ABSTRACT
Most communication in the modern era takes place over some type of cyber network, to include telecommunications, banking, public utilities, and health systems. Information gained from illegitimate network access can be used to create catastrophic effects at the individual, corporate, national, and even international levels, making cyber security a top priority. Cyber networks frequently encounter amounts of network traffic too large to process real-time threat detection efficiently. Reducing the amount of information necessary for a network monitor to determine the presence of a threat would likely aide in keeping networks more secure. This thesis uses network traffic data captured during the Department of Defense Cyber Defense Exercise to determine which features of network traffic are salient to detecting and classifying threats. After generating a set of 248 features from the capture data, feed-forward artificial neural networks were generated and signal-to-noise ratios were used to prune the feature set to 18 features while still achieving an accuracy ranging from 83% - 94%. The salient features primarily come from the transport layer section of the network traffic data and involve the client/server connection parameters, size of the initial data sent, and number of segments and/or bytes sent in the flow.

15. SUBJECT TERMS
Network Threat Detection; Feature Selection, Neural Networks, Flow Feature Generation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kenneth W. Bauer, Jr.
U	U	U	UU	146	19b. TELEPHONE NUMBER (Include Area Code) (937)255-3636, ext 4328

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18